

# ACM 国际大学生程序设计 竞赛题解 (2)

赵端阳 袁 鹤 编著

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

## 内 容 简 介

随着各大专院校参加 ACM/ICPC 热情的高涨,迫切需要有有关介绍 ACM 国际大学生程序设计竞赛题解的书籍。本书根据浙江大学在线题库的部分题目,经过分类、筛选、汇编,并进行了解答(个别特别简单或者特别复杂的题目未选择),比较详细地分析和深入浅出地讲解了解题的方法和用到的算法。题目的类型包括基础编程、模拟、字符串处理、搜索、动态规划、回溯、图论、几何和数学题。

本书可以作为高等院校有关专业的本科和大专学生参加国际大学生程序设计竞赛的辅导教材,或者作为高等院校数据结构、C/C++程序设计或算法设计与分析等相关课程的教学参考书。

各题目的源代码及相关资料可在 <http://www.hxedu.com.cn> 下载。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究。

## 图书在版编目(CIP)数据

ACM 国际大学生程序设计竞赛题解. 2 / 赵端阳, 袁鹤编著. —北京: 电子工业出版社, 2010.7

ISBN 978-7-121-11171-6

I. ①A… II. ①赵… ②袁… III. ①程序设计—竞赛—高等学校—解题 IV. ①TP311.1-44

中国版本图书馆 CIP 数据核字(2010)第 116153 号

责任编辑: 陈晓莉

印 刷:

装 订:

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本: 787×1092 1/16 印张: 22.25 字数: 570 千字

印 次: 2010 年 3 月第 1 次印刷

印 数: 4000 册 定价: 39.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888。

质量投诉请发邮件至 [zltz@phei.com.cn](mailto:zltz@phei.com.cn), 盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线:(010) 88258888。

# 前 言

ACM 国际大学生程序设计竞赛 (ACM/ICPC: ACM International Collegiate Programming Contest) 是由国际计算机界历史悠久、颇具权威性的组织 ACM 学会 (Association for Computing Machinery, 美国计算机协会) 主办, 是世界上公认的规模最大、水平最高的国际大学生程序设计竞赛, 其目的旨在使大学生运用计算机来充分展示自己分析问题和解决问题的能力。1970 年在美国 Texas a&m 大学举办了首次区域竞赛, 从而拉开了国际大学生程序设计竞赛的序幕。该项竞赛从 1970 年举办至今已历 29 届, 因历届竞赛都荟萃了世界各大洲的精英, 云集了计算机界的“希望之星”, 而受到国际各知名大学的重视, 并受到全世界各著名计算机公司的高度关注, 成为世界各国大学生最具影响力的国际级计算机类的赛事。

此项赛事的主办目的不单是培养参赛选手的创造力, 团队合作精神以及他们在软件程序开发过程中的创新意识, 同时也是检测选手们在压力下进行开发活动的的能力。因此, ACM 国际大学生程序设计竞赛是参赛选手展示计算机才华的广阔舞台, 是著名大学计算机教育成果的直接体现, 是信息企业与世界顶尖计算机人才对话的最好机会。该项竞赛分区域预赛和国际决赛两个阶段进行, 各预赛区第一名自动获得参加世界决赛的资格, 世界决赛安排在每年的 3~4 月举行, 而区域预赛安排在上一年度的 9~12 月在各大洲举行。从 1998 年开始, IBM 公司连续赞助该项赛事的世界决赛和区域预赛。这项比赛是以大学为单位组队 (每支队伍由教练、3 名正式队员, 一名后备队员组成) 参赛。

国际 ACM 比赛 1996 年才进入中国内地, 上海交通大学作为我国内地高校最早的参赛队之一, 曾 7 次进军总决赛, 并于 2002 年将 ACM 金杯首次带到亚洲。打破了几十年来欧美国对这一赛事绝对的统治地位, 更震惊了世界。

国际 ACM 比赛是世界上规模最大、历史最长、影响最深的全球性计算机专业竞赛, 它要求每一名队员不仅具有扎实的数学功底、非凡的算法设计能力、娴熟的编程技巧, 而且必须具备很好的协作精神、稳定的心理素质、快速的临场应变能力。

为了帮助各个大专院校的大学生们了解国际大学生程序设计竞赛, 了解其程序设计的方法, 提高参与校级、省级和亚洲区赛国际大学生程序设计竞赛的兴趣, 特编写这本题解。

全书共分九章:

第 1 章, 基础编程题。主要是比较容易的题目, 也称简单题, 尤其适合刚刚开始熟悉 ACM 大学生程序设计竞赛做题的同学。通过这些题目的练习, 主要是熟悉数据的输入/输出格式, 基本编程方法, 在线提交系统的使用, 常见错误及其对策。

第 2 章, 模拟算法题。根据题目的要求逐步实现目标, 虽然没有经典的算法可以使用, 正确理解题目是关键的因素。例如题目“Robbery”和“Tournament Seeding”等。

第 3 章, 字符串处理题。主要是字符串的处理, 这也是考察参赛者细心的一类题目, 需要对各种情况进行仔细的分析, 予以正确的处理。例如题目“Language of FatmMouse”和“Bio-Informatics”等。

第 4 章, 基本数据结构题。常见的有二叉树、堆栈和列表等, 例如题目“Matrix Chain

Multiplication”和“Code the Tree”等。

第5章，搜索算法题。这是一个最常见的类型，通常有深度优先搜索和广度优先搜索算法。例如题目“Channel Allocation”和“Gamblers”等。

第6章，动态规划算法题。这些题目使用动态规划算法，会获得较快的运行时间和效率，例如题目“Monkey and Banana”和“FatMouse and Cheese”等。

第7章，回溯算法题。这些题目使用回溯算法，会获得较快的运行时间和效率，例如题目“Dreisam Equations”等。

第8章，图论算法题。主要包含拓扑排序、Floyd 算法和二分图等，例如题目“Stockbroker Grapevine”和“Ouroboros Snake”等。

第9章，几何和数学题。主要是几何题和一些数学计算题，例如题目“Subway”和“Equidistance”等。

本部分通过大量的实例介绍了竞赛中常用的算法，并对如何灵活地应用这些算法进行了比较详细地分析和深入浅出地讲解。书中所用的语言是 C/C++，并在 MinGW Developer Studio 中调试通过。在运行时间和内存占用的性能方面，C++并不比 C 语言优越多少，只是 C++有丰富的模板库函数，输入/输出语句简单，编写的代码看起来格式更工整而已。在有大量输入/输出数据的题目中，书中会特别提醒读者需要使用 C 语言的输入/输出。显然，“Accepted”是我们的目标，算法是我们不断探索的道路，好的算法让我们容易达到目标。书中介绍的算法，虽然作者经过反复斟酌，不断优化和简化，但仍然不能认为是最优的。因为对算法的探索，正如金庸笔下的大侠们苦练武功一样，是没有止境的。

本书中虽然描述的是 ACM 国际大学生程序设计竞赛中最基本的算法，但它也已经超出了一般本科教科书所讲授的范围。清华大学计算机科学与技术系博士生导师、国际信息学奥林匹克中国队总教练吴文虎教授认为算法的确是艺术，“艺术与科学是相通的，都会给人以美的享受。”并感叹道：“体味思维艺术之美，我以为这可能是更高层次的享受”<sup>[1]</sup>。希望读者能从本书中体会到这一点。

本书可以作为高等院校有关专业的本科和大专学生参加国际大学生程序设计竞赛的辅导教材，或者作为高等院校数据结构、C/C++ 程序设计或算法设计与分析等相关课程的教学参考书，旨在培养和提高学生参加 ACM 国际大学生程序设计竞赛的兴趣。

本书在编写过程中得到了浙江大学在线裁判系统管理员的大力支持，在此表示非常感谢。

感谢沈仙桥同学翻译了大部分的题目。同时也感谢 ACM 参赛队员戚勇、林刚、王海江、沈懿华、郑彦良、戴俊、周智栋、应建伟等同学的热情帮助。

由于作者水平所限，书中难免有不足之处，恳请广大读者批评指正。作者电子邮件地址：727946579@qq.com 和 hzyuanhe@163.com。

作 者  
2010 年 6 月于杭州

---

[1] 吴文虎主编，孙贺编著，序，程序设计中的组合数学，清华大学出版社，2005 年 5 月

# 目 录

第一章 基础编程题	1
ZJU1086-Octal Fractions	1
ZJU1089-Lotto	3
ZJU1090-The Circumference of the Circle	6
ZJU1095-Humble Numbers	8
ZJU1099-HTML	11
ZJU1105-FatMouse's Tour	15
ZJU1115-Digital Roots	17
ZJU1122-Clock	19
ZJU1139-Rectangles	22
ZJU1151-Word Reversal	25
ZJU1152-A Mathematical Curiosity	27
ZJU1154-Niven Numbers	29
第二章 模拟算法题	32
ZJU1088-System Overload	32
ZJU1098-Simple Computers	36
ZJU1121-Reserve Bookshelf	40
ZJU1143-Date Bugs	48
ZJU1144-Robbery	51
ZJU1146-LC-Display	56
ZJU1153-Tournament Seeding	61
ZJU1160-Biorhythms	65
第三章 字符串处理题	70
ZJU1109-Language of FatMouse	70
ZJU1111-Poker Hands	74
ZJU1116-A Well-Formed Problem	81
ZJU1126-Bio-Informatics	88
ZJU1159-487-3279	93
第四章 基本数据结构题	99
ZJU1094-Matrix Chain Multiplication	99
ZJU1097-Code the Tree	104
ZJU1156-Unscrambling Images	109
第五章 搜索算法题	116
ZJU1084-Channel Allocation	116
ZJU1085-Alien Security	120

ZJU1091-Knight Moves .....	126
ZJU1101-Gamblers .....	132
ZJU1103-Hike on a Graph .....	135
ZJU1129-Erdos Numbers .....	141
ZJU1136-Multiple .....	147
ZJU1142-Maze .....	152
ZJU1148-The Game .....	158
<b>第六章 动态规划算法题 .....</b>	<b>163</b>
ZJU1093-Monkey and Banana .....	163
ZJU1100-Mondriaan's Dream .....	169
ZJU1102-Phylogenetic Trees Inherited .....	174
ZJU1107-FatMouse and Cheese .....	180
ZJU1108-FatMouse's Speed .....	183
ZJU1132-Railroad .....	188
ZJU1147-Formatting Text .....	195
ZJU1149-Dividing .....	201
<b>第七章 回溯算法题 .....</b>	<b>207</b>
ZJU1145-Dreisam Equations .....	207
ZJU1157-A Plug for UNIX .....	214
<b>第八章 图论算法题 .....</b>	<b>223</b>
ZJU1082-Stockbroker Grapevine .....	223
ZJU1083-Frame Stacking .....	228
ZJU1092-Arbitrage .....	235
ZJU1117-Entropy .....	239
ZJU1118-N-Credible Mazes .....	245
ZJU1119-SPF .....	251
ZJU1127-Roman Forts .....	256
ZJU1130-Ouroboros Snake .....	262
ZJU1134-Strategic Game .....	268
ZJU1137-Girls and Boys .....	273
ZJU1140-Courses .....	278
ZJU1141-Closest Common Ancestors .....	281
ZJU1150-S-Trees .....	285
<b>第九章 几何和数学题 .....</b>	<b>291</b>
ZJU1081-Points Within .....	291
ZJU1096-Subway .....	296
ZJU1104-Leaps Tall Buildings .....	301
ZJU1110-Dick and Jane .....	306
ZJU1112-Equidistance .....	309
ZJU1114-Problem Bee .....	315

ZJU1123-Triangle Encapsulation .....	319
ZJU1125-Floating Point Numbers.....	325
ZJU1128-Atlantis.....	330
ZJU1133-Smith Numbers .....	335
ZJU1158-Treasure Hunt.....	339
索引 .....	346
参考文献 .....	348

# 第一章 基础编程题

在本章的题目大部分都比较简单，作为刚刚开始参加竞赛的练习题。

## ZJU1086-Octal Fractions<sup>[1、2、3]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768KB

---

Fractions in octal (base 8) notation can be expressed exactly in decimal notation. For example, 0.75 in octal is 0.953125 ( $7/8 + 5/64$ ) in decimal. All octal numbers of  $n$  digits to the right of the octal point can be expressed in no more than  $3n$  decimal digits to the right of the decimal point.

Write a program to convert octal numerals between 0 and 1, inclusive, into equivalent decimal numerals. The input to your program will consist of octal numbers, one per line, to be converted. Each input number has the form  $0.d_1d_2d_3 \cdots d_k$ , where the  $d_i$  are octal digits (0..7). There is no limit on  $k$ . Your output will consist of a sequence of lines of the form

$$0.d_1d_2d_3 \cdots d_k [8] = 0.D_1D_2D_3 \cdots D_m [10]$$

where the left side is the input (in octal), and the right hand side the decimal (base 10) equivalent. There must be no trailing zeros, i.e.  $D_m$  is not equal to 0.

### SAMPLE INPUT

```
0.75
0.0001
0.01234567
```

### SAMPLE OUTPUT

```
0.75 [8] = 0.953125 [10]
0.0001 [8] = 0.000244140625 [10]
0.01234567 [8] = 0.020408093929290771484375 [10]
```

### Problem Source:

South Africa 2001

### 【题目大意】

八进制表示的小数可用十进制数精确地表示。例如，八进制数 0.75 等于十进制数 0.953125

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1086>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1131>

[3] <http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2245>



( $7/8+5/64$ )。八进制小数点右边的  $n$  位数可表示成小数位数不多于  $3n$  位的十进制数。

**编程任务：**将  $0\sim 1$  之间的八进制小数，转换为等值的十进制数。程序的输入是八进制数，每行一个，分别转换成十进制数。输入数据的格式为  $0.d_1d_2d_3\cdots d_k$ ，其中  $d_i$  为八进制数值 ( $0\cdots 7$ )， $k$  值没有限制。程序输出格式：

$$0.d_1d_2d_3\cdots d_k [8] = 0.D_1D_2D_3\cdots D_m [10]$$

左边是输入的八进制数，而右边是等值的十进制数。小数后面没有拖零，也就是  $D_m$  不为 0。

### 【算法分析】

本题需要将八进制表示的小数用十进制数精确地表示，由于位数很多，需要采用高精度的方法计算。用数学公式表达这种转换关系：

$$(0.75)_8 = (7\times 8^{-1} + 5\times 8^{-2})_{10} = (0.93125)_{10}$$

被除数和除数都要使用高精度运算，编程是比较麻烦的。改变一下计算公式：

$$(0.75)_8 = (7\times 8^{-1} + 5\times 8^{-2})_{10} = ((5/8+7)/8)_{10} = (0.93125)_{10}$$

也就是采用累除的方法，计算起来就很方便。从八进制小数的最低位开始，除以 8 后，与其前一位相加，一直到小数点后的第一位小数。如上例数据，只有两个小数位，分两步运算：

①  $5/8=0.615$

②  $(0.625+7)/8=0.93125$

这样，我们只要实现除以 8 的高精度运算。

### 【程序代码】

---

程序名称： zju1086.c  
题    目：  Octal Fractions  
提交语言：  C  
运行时间：  00:00.00  
运行内存：  388KB

---

```
#include <stdio.h>
#include <string.h>
#define MaxN 100
int main()
{
    char src[MaxN];                //八进制的小数
    int i, j;
    while(scanf("%s",src) != EOF)
    {
        char dest[MaxN] = {'0'};  //十进制的小数
        int index = 0;             //十进制小数的长度
        //从八进制小数的最后一位开始累除
        for(i=strlen(src)-1; i>1; i--)
        {
            int num = src[i] - '0'; //八进制小数的当前位
            //实现除以 8 的高精度运算
            int temp;
            //最后的余数 num 也必须除尽
```

```

        for(j=0; j<index || num; j++)
        {
            temp = num*10 + (j<index?dest[j]-'0':0);
            dest[j] = temp/8+'0';          //商
            num = temp%8;                  //余数
        }
        index = j;                        //十进制小数的实际位数
    }
    dest[j] = '\0';                       //字符串结束标志
    printf("%s [8] = 0.%s [10]\n", src, dest);
}
return 0;
}

```

## ZJU1089-Lotto<sup>[1, 2, 3]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768KB

---

In a Lotto I have ever played, one has to select 6 numbers from the set  $\{1, 2, \dots, 49\}$ . A popular strategy to play Lotto — although it doesn't increase your chance of winning — is to select a subset  $S$  containing  $k$  ( $k > 6$ ) of these 49 numbers, and then play several games with choosing numbers only from  $S$ . For example, for  $k=8$  and  $S=\{1, 2, 3, 5, 8, 13, 21, 34\}$  there are 28 possible games:  $[1, 2, 3, 5, 8, 13]$ ,  $[1, 2, 3, 5, 8, 21]$ ,  $[1, 2, 3, 5, 8, 34]$ ,  $[1, 2, 3, 5, 13, 21]$ ,  $\dots$ ,  $[3, 5, 8, 13, 21, 34]$ .

Your job is to write a program that reads in the number  $k$  and the set  $S$  and then prints all possible games choosing numbers only from  $S$ .

### Input Specification

The input file will contain one or more test cases. Each test case consists of one line containing several integers separated from each other by spaces. The first integer on the line will be the number  $k$  ( $6 < k < 13$ ). Then  $k$  integers, specifying the set  $S$ , will follow in ascending order. Input will be terminated by a value of zero (0) for  $k$ .

### Output Specification

For each test case, print all possible games, each game on one line. The numbers of each game have to be sorted in ascending order and separated from each other by exactly one space. The games themselves have to be sorted lexicographically, that means sorted by the lowest number first, then by the second lowest and so on, as demonstrated in the sample output below. The test cases have to be

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1089>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2245>

[3] <http://acm.uva.es/p/v4/441.html>

separated from each other by exactly one blank line. Do not put a blank line after the last test case.

## Sample Input

```
7 1 2 3 4 5 6 7
8 1 2 3 5 8 13 21 34
0
```

## Sample Output

1 2 3 4 5 6	1 2 3 5 8 13	1 2 8 13 21 34
1 2 3 4 5 7	1 2 3 5 8 21	1 3 5 8 13 21
1 2 3 4 6 7	1 2 3 5 8 34	1 3 5 8 13 34
1 2 3 5 6 7	1 2 3 5 13 21	1 3 5 8 21 34
1 2 4 5 6 7	1 2 3 5 13 34	1 3 5 13 21 34
1 3 4 5 6 7	1 2 3 5 21 34	1 3 8 13 21 34
2 3 4 5 6 7	1 2 3 8 13 21	1 5 8 13 21 34
	1 2 3 8 13 34	2 3 5 8 13 21
	1 2 3 8 21 34	2 3 5 8 13 34
	1 2 3 13 21 34	2 3 5 8 21 34
	1 2 5 8 13 21	2 3 5 13 21 34
	1 2 5 8 13 34	2 3 8 13 21 34
	1 2 5 8 21 34	2 5 8 13 21 34
	1 2 5 13 21 34	3 5 8 13 21 34

## Problem Source

University of Ulm Local Contest 1996

### 【题目大意】

在我曾经玩过的一种对号码的纸牌游戏（乐透）里，玩家必须从 $\{1, 2, \dots, 49\}$ 中选 6 个数。玩 Lotto 的一个流行的策略是（虽然它并不增加你赢的机会）：就是从这 49 个数中，选出  $k$  ( $k > 6$ ) 个数组成一个子集  $S$ ，然后只从  $S$  里拿出牌来玩几局游戏。例如， $k=8$ ,  $S=\{1, 2, 3, 5, 8, 13, 21, 34\}$ ，那么有 28 场可能的游戏： $[1, 2, 3, 5, 8, 13]$ ,  $[1, 2, 3, 5, 8, 21]$ ,  $[1, 2, 3, 5, 8, 34]$ ,  $[1, 2, 3, 5, 13, 21]$ ,  $\dots$ ,  $[3, 5, 8, 13, 21, 34]$ 。

### 编程任务

读取数字  $k$  和一组数  $S$ ，输出由  $S$  中的数组成的所有可能的游戏。

### 输入格式

输入有一组或多组测试数据。每组测试数据一行，数与数之间用空格隔开。每行第一个整数是  $k$  ( $6 < k < 13$ )，然后是从小到排列的  $k$  个整数，即子集  $S$ 。当  $k$  为零时，输入结束。

### 输出格式

对于每组测试数据，输出所有可能的游戏，每个游戏占一行。游戏里的数字按升序排列，数与数之间有一个空格。所有游戏也要按字典序排列，也就是，首先比较各局游戏的第一个数的大小，然后是第二个数的大小，依次类推，如下面的输出样例所示。不同的测试例之间用一个空行隔开。最后一组测试例之后没有空行。

## 【算法分析】

本题看起来很复杂：它是要实现从一组数里挑选 6 个数的所有游戏组合，输出时每个游戏里的数字按升序排列，所有的游戏还要按字典序排列。所以在网上能够看到多种实现算法，如深度优先搜索等。

最简单的办法是模拟。因为数字已经有序，所以就从前往后挑选，这样挑选出来的数字，不仅每个游戏里的数字是升序的，而且所有游戏是字典序的。如图 1-1 所示，挑选第一个数字：

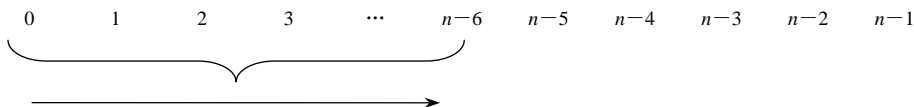


图 1-1 挑选第一个数字的示意图

依次在第 0~( $n-6$ ) 个数字之间，挑选第一个数字，序号为  $a$ 。

第二个数字是第一个数字的序号  $a$  加 1，依次挑选数字，一直挑选到第  $n-5$  个数字，序号为  $b$ 。依此类推。

## 【程序代码】

程序名称:	zju1089.c
题 目:	Lotto
提交语言:	C++
运行时间:	00:00.00
运行内存:	392KB

```
#include<stdio.h>
int number[14];                                //存放原始数据

int main(){
    int line = 0;                               //空行标志
    int n;                                       //数字的个数
    while(scanf("%d", &n) && n) {
        if (line) printf("\n");
        for(int i=0; i<n; ++i)                 //读取子集 S
            scanf("%d", &number[i]);
        //模拟挑选数字，并输出结果
        for(int a = 0; a<n-5; ++a)
            for(int b=a+1; b<n-4; ++b)
                for(int c=b+1; c<n-3; ++c)
                    for(int d=c+1; d<n-2; ++d)
                        for(int e=d+1; e<n-1; ++e)
                            for(int f=e+1; f<n; ++f)
                                printf("%d %d %d %d %d %d\n",
                                    number[a],number[b],number[c],
                                    number[d],number[e],number[f]);
        line++;
    }
```

```

        line = 1;
    }
}

```

## ZJU1090-The Circumference of the Circle<sup>[1, 2, 3]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768KB

---

To calculate the circumference of a circle seems to be an easy task — provided you know its diameter. But what if you don't? You are given the cartesian coordinates of three non—collinear points in the plane. Your job is to calculate the circumference of the unique circle that intersects all three points.

### Input Specification

The input file will contain one or more test cases. Each test case consists of one line containing six real numbers  $x_1, y_1, x_2, y_2, x_3, y_3$ , representing the coordinates of the three points. The diameter of the circle determined by the three points will never exceed a million. Input is terminated by end of file.

### Output Specification

For each test case, print one line containing one real number telling the circumference of the circle determined by the three points. The circumference is to be printed accurately rounded to two decimals. The value of  $\pi$  is approximately 3.141592653589793.

### Sample Input

```

0.0 -0.5 0.5 0.0 0.0 0.5
0.0 0.0 0.0 1.0 1.0 1.0
5.0 5.0 5.0 7.0 4.0 6.0
0.0 0.0 -1.0 7.0 7.0 7.0
50.0 50.0 50.0 70.0 40.0 60.0
0.0 0.0 10.0 0.0 20.0 1.0
0.0 -500000.0 500000.0 0.0 0.0 500000.0

```

### Sample Output

```

3.14
4.44
6.28
31.42
62.83

```

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1090>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2242>

[3] <http://acm.uva.es/p/v4/438.html>

632.24  
3141592.65

## Problem Source

University of Ulm Local Contest 1996

### 【题目大意】

假设你知道了圆的直径，计算其周长似乎是一件容易的事情。但是如果你又不会计算呢？在平面笛卡儿坐标系内，输入三个不在同一直线上的点。经过三个坐标点的圆只有一个，你的任务是求出该圆的周长。

### 输入格式

输入包含一组或多组测试数据。每个测试例一行，有 6 个实数  $x_1, y_1, x_2, y_2, x_3, y_3$ ，分别表示三个点的坐标。由这三个坐标点决定的圆直径不会超过  $10^6$ 。输入以文件结束符为标志。

### 输出格式

对于每组测试例，输出一行，是一个实数，由三个坐标点构成的圆周长。圆周长精确到百分位。 $\pi$ 的近似值是 3.141592653589793。

### 【算法分析】

本题是已知三点的坐标，求外接圆的圆周长。

设圆心坐标是  $(x_0, y_0)$ ，半径是  $r$ ，则

$$(x-x_0)^2 + (y-y_0)^2 = r^2$$

将已知的三点坐标代入，求出半径  $r$ ，即可求出圆周长。但是解这个方程有点麻烦，利用三点构成的三角形，可以根据公式求解。

假设三角形的三边长分别是： $a, b, c$ ，对应角分别是  $A, B, C$ ，面积为  $S$ ，如图 1-2 所示。

(1) 三角函数求解半径  $r$

根据余弦定理：

$$c^2 = a^2 + b^2 - 2ab \cos C$$

根据正弦定理：

$$\frac{c}{\sin C} = 2r$$

$$\because \sin^2 C + \cos^2 C = 1$$

$$\therefore \left(\frac{c}{2r}\right)^2 + \left(\frac{a^2 + b^2 - c^2}{2ab}\right)^2 = 1$$

从中解出  $r$  即可。

(2) 面积公式求解半径  $r$

三角形的外接圆半径：

$$r = \frac{abc}{4S}$$

根据海伦公式计算面积：

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

其中： $p = (a+b+c)/2$ 。

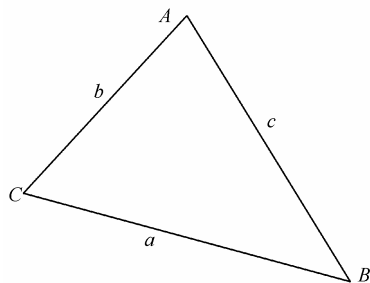


图 1-2 三角形的边角关系

## 【程序代码】

---

程序名称:      zju1090.c  
题    目:      The Circumference of the Circle  
提交语言:      C  
运行时间:      00:00.00  
运行内存:      396K

---

```
#include <stdio.h>
#include <math.h>

#define PI 3.141592653589793
double square(double x){
    return x*x;
}

int main()
{
    double x1, y1, x2, y2, x3, y3;           //三点坐标
    double a2, b2, c2;                       //三边长的平方
    double r;                                //半径
    while(scanf("%lf%lf%lf%lf%lf%lf", &x1, &y1, &x2, &y2, &x3, &y3) != EOF)
    {
        //根据三角函数求解半径 r
        a2 = square(x1-x2) + square(y1-y2);
        b2 = square(x1-x3) + square(y1-y3);
        c2 = square(x3-x2) + square(y3-y2);
        r = sqrt(c2 / (1-square(a2+b2-c2)/a2/b2/4)) / 2;
        printf("%.2lf\n", 2*PI*r);
    }
    return 0;
}
```

## ZJU1095-Humble Numbers<sup>[1、2、3]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768K

---

A number whose only prime factors are 2,3,5 or 7 is called a humble number. The sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 24, 25, 27, ... shows the first 20 humble numbers. Write a program to find and print the  $n$  element in this sequence.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1095>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2247>

[3] <http://acm.uva.es/p/v4/443.html>

## Input Specification

The input consists of one or more test cases. Each test case consists of one integer  $n$  with  $1 \leq n \leq 5842$ . Input is terminated by a value of zero (0) for  $n$ .

## Output Specification

For each test case, print one line saying "The  $n$ th humble number is number.". Depending on the value of  $n$ , the correct suffix "st", "nd", "rd", or "th" for the ordinal number  $n$ th has to be used like it is shown in the sample output.

### Sample Input

```
1
2
3
4
11
12
13
21
22
23
100
1000
5842
0
```

### Sample Output

```
The 1st humble number is 1.
The 2nd humble number is 2.
The 3rd humble number is 3.
The 4th humble number is 4.
The 11th humble number is 12.
The 12th humble number is 14.
The 13th humble number is 15.
The 21st humble number is 28.
The 22nd humble number is 30.
The 23rd humble number is 32.
The 100th humble number is 450.
The 1000th humble number is 385875.
The 5842nd humble number is 2000000000.
```

## Problem Source

University of Ulm Local Contest 1996

### 【题目大意】

质因数是 2, 3, 5 或者 7 的数称为丑数 (humble number)。1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 24, 25, 27, ... 是头 20 个 humble number。

写一个程序，输出上面序列中的第  $n$  个元素。

### 输入格式

输入一组或多组测试数据。每组测试数据一个整数  $n$  ( $1 \leq n \leq 5842$ )。输入  $n$  零 (0) 结束。

### 输出格式

对于每组测试数据,输出 "The  $n$ th humble number is number.". 根据值  $n$ , 后缀有 "st", "nd", "rd" 或 "th", 如下输出所示。

### 【算法分析】

本题是根据给定的数  $n$ , 计算相应的丑数 (humble number)。在 University of Ulm Local



Contest 1996 的竞赛资料页面<sup>[1]</sup>上，给出了一个标程。主要是采用打表的办法。

计算丑数是一个非常耗时的过程，如果对每一个  $n$ ，都去计算相应的丑数，很容易超时，所以在网上有很多加速的办法，算法都比较复杂。最简单并且速度最快的办法，就是标程提供的方法：打表，将 5842 个丑数一次性计算出来，放到数组  $a$  中：

```
int a[5850];
```

然后，每读取一个  $n$ ，直接查表输出就可以了，速度自然很快。

打表算法：

一个丑数可能由 2, 3, 5 和 7 中的一个或者多个构成，分别对它们设一个计数器：

```
int p2, p3, p5, p7;
```

采用递推的办法，从 2 开始计算。每一次分别计算： $2 \times a[p2]$ ， $3 \times a[p3]$ ， $5 \times a[p5]$ ， $7 \times a[p7]$ ，从中取最小的数作为当前的丑数；然后判断这个最小的数是由  $2 \times a[p2]$ ， $3 \times a[p3]$ ， $5 \times a[p5]$ ， $7 \times a[p7]$  中哪一个产生的，将所在的计数器加 1。其过程如表 1-1 所示。

表 1-1 递推过程

$n$	$2 \times a[p2]$	$3 \times a[p3]$	$5 \times a[p5]$	$7 \times a[p7]$	$p2$	$p3$	$p5$	$p7$	$a[n]$	相应 指针
1					1	1	1	1	1	
2	2	3	5	7	2	1	1	1	2	$p2++$
3	4	3	5	7	2	2	1	1	3	$p3++$
4	4	6	5	7	3	2	1	1	4	$p2++$
5	6	6	5	7	3	2	2	1	5	$p5++$
6	6	6	10	7	4	3	2	1	6	$p2++$
7	8	9	10	7	4	3	2	2	7	$p7++$
8	8	9	10	14	5	3	2	2	8	$p2++$
9	10	9	10	14	5	4	2	2	9	$p3++$
...										

这种方法既能枚举所有的丑数，而且所产生的丑数是单调上升的。

### 【程序代码】

---

程序名称： zju1095.c  
题    目：  Humble Numbers  
提交语言：  C  
运行时间：  00:00.01  
运行内存：  416KB

---

```
#include <stdio.h>

#define min(a,b) ((a)<(b)?(a):(b))
#define min4(a,b,c,d) min(min(a,b),min(c,d))
```

---

[1] <http://www.informatik.uni-ulm.de/acm/Locals/1996/>

```

int a[5850];                                //存放丑数

int main()
{
    int n = 1;
    int p2, p3, p5, p7;                    //2, 3, 5 和 7 的计数器
    p2 = p3 = p5 = p7 = 1;                //初值为 1
    a[1] = 1;
    //从 2 开始递推计算，一共 5842 个丑数
    while (a[n]<2000000000)
    {
        a[++n] = min4(2*a[p2],3*a[p3],5*a[p5],7*a[p7]);    //取最小值
        //相应的计数器加 1
        if (a[n]==2*a[p2]) p2++;
        if (a[n]==3*a[p3]) p3++;
        if (a[n]==5*a[p5]) p5++;
        if (a[n]==7*a[p7]) p7++;
    }
    while (scanf("%d",&n) && n)
    {
        printf("The %d",n);
        //下面是处理序数词，变量 ten 是第十位数
        int ten = n/10%10;
        //当十位数不为 1，而尾数是 1, 2 和 3 时，分别输出"st", "nd", "rd"
        if (n%10==1 && ten!=1) printf("st");
        else if (n%10==2 && ten!=1) printf("nd");
        else if (n%10==3 && ten!=1) printf("rd");
        else printf("th");                    //其余数字
        printf(" humble number is %d.\n",a[n]);
    }
    return 0;
}

```

## ZJU1099-HTML<sup>[1、2]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768KB

---

If you ever tried to read a html document on a Macintosh, you know how hard it is if no Netscape is installed.

Now, who can forget to install a HTML browser? This is very easy because most of the times

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1099>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2271>

you don't need one on a MAC because there is a Acrobat Reader which is native to MAC. But if you ever need one, what do you do?

Your task is to write a small html—browser. It should only display the content of the input—file and knows only the html commands (tags) `<br>` which is a linebreak and `<hr>` which is a horizontal ruler. Then you should treat all tabulators, spaces and newlines as one space and display the resulting text with no more than 80 characters on a line.

## Input Specification

The input consists of a text you should display. This text consists of words and HTML tags separated by one or more spaces, tabulators or newlines.

A word is a sequence of letters, numbers and punctuation. For example, "abc,123" is one word, but "abc, 123" are two words, namely "abc," and "123". A word is always shorter than 81 characters and does not contain any '`<`' or '`>`'. All HTML tags are either `<br>` or `<hr>`.

## Output Specification

You should display the resulting text using this rules:

- If you read a word in the input and the resulting line does not get longer than 80 chars, print it, else print it on a new line.
- If you read a `<br>` in the input, start a new line.
- If you read a `<hr>` in the input, start a new line unless you already are at the beginning of a line, display 80 characters of '—' and start a new line (again).

The last line is ended by a newline character.

## Sample Input

```
Hallo, dies ist eine
ziemlich lange Zeile, die in Html
aber nicht umgebrochen wird.
<br>
Zwei <br> <br> produzieren zwei Newlines.
Es gibt auch noch das tag <hr> was einen Trenner darstellt.
Zwei <hr> <hr> produzieren zwei Horizontal Rulers.
Achtung mehrere Leerzeichen irritieren

Html genauso wenig wie

mehrere Leerzeilen.
```

## Sample Output

```
Hallo, dies ist eine ziemlich lange Zeile, die in Html aber nicht umgebrochen
wird.
Zwei
```

produzieren zwei Newlines. Es gibt auch noch das tag

-----  
was einen Trenner darstellt. Zwei  
-----

-----  
produzieren zwei Horizontal Rulers. Achtung mehrere Leerzeichen irritieren Html  
genauso wenig wie mehrere Leerzeilen.

## Problem Source

University of Ulm Local Contest 1999

### 【题目大意】

如果你曾经试着在没有安装网景浏览器的个人计算机上阅读一个 html 文档，你知道这是多么困难。

现在，谁能忘记安装一个 HTML 浏览器呢？对于 MAC 机器，这是非常容易的。因为大部分时间里你并不需要浏览器，MAC 机器本身就有有一个 Acrobat 阅读器。但倘若你需要浏览器，你将怎么办？

### 编程任务

编写一个迷你 html 浏览器。浏览器只显示输入文件的内容，并且只需要识别 html 标记：  
<br>是换行，<hr>是水平线。程序应该将所有的制表符、空格、换行符当作一个空格，每行输出不超过 80 个字符。

### 输入格式

输入是浏览器需要显示的文本。文件包含词和 HTML 标记，中间有一个或多个空格、制表符和换行符。

词是一串字母、数字和标点。例如，“abc,123”是一个词，但是“abc, 123”是两个词，也就是“abc”和“123”。每个词不超过 81 个字符且不包含“<”或“>”。HTML 标签只有<br>或<hr>。

### 输出格式

按下列规则显示文档：

- 对输入的一个单词，如果在当前行输出时不超过 80 个字符，那么就直接输出，否则在下一行输出。
- 如果输入的是<br>，则换行。
- 如果输入的是<hr>，则换行，如果恰好位于新行的开头就不必换行，输出 80 个“-”字符，再换行。

最后是一个空行。

### 【算法分析】

这是一道模拟题，根据题目的要求进行模拟。

数据结构：

累计一行内已经输出的字符个数为

```
int count;
每个单词为
char word[81];
```

## 【程序代码】

---

程序名称: zju1099.c

题 目: HTML

提交语言: C

运行时间: 00:00.00

运行内存: 160KB

---

```
#include <stdio.h>
#include <string.h>

int main()
{
    int count=0;                //一行内已经输出的字符个数
    char word[81];              //单词
    while (scanf("%s",word)!=EOF)
    {
        //处理<hr>标记
        if (strcmp(word,"<hr>")==0)
        {
            //如果不是新行, 则回车
            if (count!=0) printf("\n");
            int i;
            for (i=0; i<80; i++) printf("-");
            printf("\n");
            //清除计数器
            count=0;
        }
        //处理<br>标记
    }else if (strcmp(word,"<br>")==0)
    {    //回车并清除计数器
        printf("\n");
        count=0;
    }else{
        //输出当前单词会超过 80 个字符
        if (count+strlen(word)+(count==0?0:1)>80)
        {
            printf("\n%s", word);                //在下一行输出
            count=strlen(word);
        }else{
            //该单词能够在当前行输出
            if (count!=0) printf(" ");            //不是新行, 要加一个空格
            printf("%s", word);
            count+=strlen(word)+1;
        }
    }
}
```

```

        }
    }
}
printf("\n");
return 0;
}

```

## ZJU1105-FatMouse's Tour<sup>[1]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

As the Chairman of Association of Campus Mice (ACM), FatMouse is going out of his office to tour his kingdom. But with the International Cat Patrol Centers (ICPC) locating at every street, he has to carefully plan his trip so that it requires minimum time to visit everyone living at every street.

Assume that there are mice living at both sides of each street, yet FatMouse can only visit one side of a street in a single pass. He can turn any direction (including a U-turn) at any intersection, and can turn around at the end of any street. He goes at 20km/h when visiting his people on the way (shaking hands and saying "hi"), and 50km/h when passing a visited side of a street (since the cats must be awake by then).

### Input Specification

The input file contains several test cases. Each test case starts from a line with two integers: the  $x$ ,  $y$  coordinates of FatMouse's office (in metres). Up to 100 lines follow. Each gives the coordinates (in metres) of the beginning and end of a street. All streets are perfectly straight, with one side in each direction. It is possible to reach all streets from the office. Each test case is finished by a special word "java".

### Output Specification

For each test case, your output should be one line with the time, in hours and minutes, required to tour the kingdom and return to the office. Round to the nearest minute.

### Sample Input

```

0 0
0 0 10000 10000
5000 -10000 5000 10000
5000 10000 10000 10000
java

```

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1105>

# Output for Sample Input

3:55

## Problem Source

Zhejiang University Training Contest 2001

### 【题目大意】

作为 Association of Campus Mice (ACM, 校园鼠协会) 主席, FatMouse 打算走出办公室周游它的王国。但是每条街道都有 International Cat Patrol Centers (ICPC, 猫国际巡逻中心)。因此他须周密安排旅游行程, 要求以最少的时间看望住在每条街道上的每一个臣民。

假设街道两边都有鼠民居住, 然而 FatMouse 在街道一边只能看望一边的臣民。在十字路口, 它可以随意决定走向 (包括掉头), 并且在任何街道的尽头折回。途中, 以 20km/h 的速度看望它的臣民 (挥手高喊 “hi”); 对已访问过的街道, 它的速度达到 50km/h (因为猫警随时会醒来)。

### 输入格式

输入有多组测试数据。对每组测试数据, 第一行有两个整数:  $x, y$ , 是 FatMouse 办公室的坐标 (单位是 m)。接着是不超过 100 行的数据, 每行表示街道的初始位置和结束位置坐标 (单位米)。所有街道笔直延伸, 一个方向只有一边。办公室与所有街道都有交通路线。每组测试数据以单词 “java” 标志结束。

### 输出格式

对每组测试数据, 输出一行, 是以时和分表示的时间, 时间包括周游王国和回到办公室。精确到分钟。

### 【算法分析】

#### (1) FatMouse 的办公室

FatMouse 的办公室与所有街道都有交通路线, 无论它的办公室在哪里, 它总是从哪儿来回到哪儿去, 所以它的办公室具体位置, 与它周游所需要的时间无关。

#### (2) 计算周游的总路程 dist

设街道  $i$  的长度是  $distance_i$ , 则总路程 dist:

$$dist = 2 \sum_{i=1}^n distance_i$$

式中  $n$  是街道总数。

### 【程序代码】

---

程序名称:	zju1105.c
题    目:	FatMouse's Tour
提交语言:	C
运行时间:	0ms
运行内存:	160KB

---

```
#include <stdio.h>
#include <math.h>
```

```

//计算两点间的距离
double distance(double x1, double y1, double x2, double y2)
{
    return sqrt( (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) );
}

int main()
{
    double x0, y0;                //FatMouse 办公室的坐标
    double x1, y1, x2, y2;        //街道两端的坐标
    char street[80];              //街道坐标的缓存
    //对每组数据, 先读取 FatMouse 办公室的坐标 (注意有回车)
    while (scanf("%lf%lf\n", &x0, &y0)!=EOF) {
        double dist = 0;          //总的距离
        gets(street);
        while (street[0] != '\0') { //是否结束
            //从缓存中读取数据
            sscanf(street, "%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            //累加距离
            dist += distance(x1, y1, x2, y2);
            gets(street);
        }
        //将距离转换成时间
        dist = 60*2.0*dist / 20000.0;
        int spend = floor( dist + 0.5 ); //四舍五入
        printf("%dist:%02d\n", spend / 60, spend % 60);
    }
    return 0;
}

```

## ZJU1115-Digital Roots<sup>[1]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

### Background

The digital root of a positive integer is found by summing the digits of the integer. If the resulting value is a single digit then that digit is the digital root. If the resulting value contains two or more digits, those digits are summed and the process is repeated. This is continued as long as necessary to obtain a single digit.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1115>



For example, consider the positive integer 24. Adding the 2 and the 4 yields a value of 6. Since 6 is a single digit, 6 is the digital root of 24. Now consider the positive integer 39. Adding the 3 and the 9 yields 12. Since 12 is not a single digit, the process must be repeated. Adding the 1 and the 2 yields 3, a single digit and also the digital root of 39.

## Input

The input file will contain a list of positive integers, one per line. The end of the input will be indicated by an integer value of zero.

## Output

For each integer in the input, output its digital root on a separate line of the output.

Example

### Input

```
24
39
0
```

### Output

```
6
3
```

## Problem Source:

Greater New York 2000

### 【题目大意】

正整数的数根 (Digital Root) 指的是该整数各位数字的和。如果结果是一位数，那么这个数就是数根。如果结果有两个或两个以上的数字，那么只需将结果的各位数字再相加直到只是一位数字。

例如正整数 24。2 和 4 相加等于 6。因为 6 是一位数，6 就是数 24 的数根。再如正整数 39。3 和 9 相加等于 12。因为 12 不是一位数，所以必须重复该过程。把 1 和 2 相加等于 3，这是一位数，也是 39 的数根。

### 输入

输入一系列正整数，每行一个。当正整数为 0 时，输入结束。

### 输出

对输入的两个整数，输出一行，是该数的数根。

### 【算法分析】

经测试，整数位数有 1000 位，只能用字符串运算。采用字符串运算的办法，把所有数字加起来，设和为 sum。然后使用模拟的办法：

- ① 将 sum 的各位数字加起来，把结果赋给 sum；
- ② 判断 sum 是否是个位数，如果不是，转①；如果是个位数，输出该个位数，运算结束。

这里有一个技巧，数根就是  $\text{sum} \% 9$  的结果，如果  $\text{sum} \% 9 = 0$ ，数根是 9。这是因为一个整数模 9 的结果与这个整数的各位数字之和模 9 的结果相同。

### 【程序代码】

---

程序名称:      zju1115.c  
题    目:      Digital Roots  
提交语言:      C  
运行时间:      0ms  
运行内存:      160KB

---

```
#include <stdio.h>

int main() {
    char ch;
    while (1) {
        //计算该整数各位数字的和
        int sum = 0;
        while (scanf("%c", &ch) && ch!='\n') {
            sum += ch - '0';
        }
        if (sum==0) break;                //结束标志
        //模 9 运算
        if (sum%9==0) sum = 9;
        else sum = sum%9;
        printf("%d\n", sum);
    }
    return 0;
}
```

## ZJU1122-Clock<sup>[1, 2]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

You are given a standard 12-hour clock with analog display, an hour hand and a minute hand. How many times does the minute hand pass the hour hand in a given time interval?

### Sample Input

```
12 50 1 2
3 8 3 20
2 45 11 0
11 0 3 20
```

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1122>

[2] <http://acm.uva.es/archive/nuevoportal/data/problem.php?p=2077>

```

1  2 12 50
3 20  3  8

```

## Sample Output

```

Program 3 by team X
Initial time  Final time  Passes
      12:50      01:02      0
      03:08      03:20      1
      02:45      11:00      8
      11:00      03:20      4
      01:02      12:50     11
      03:20      03:08     10
End of program 3 by team X

```

The input contains an indefinite number of lines; each line contains four numbers.

The first pair of numbers represents an "initial time" the second pair represents a "final time."

In each such number pair, the first number represents hours, second number represents minutes.

The hours will be in the range 1...12, the minutes in the range 0...59.

No initial time and no final time will be an instant at which the minute hand just passes the hour hand. (In particular, 12 00 will not occur as an initial or final time.)

No initial time will be the same as the corresponding final time.

Between each initial time and corresponding final time, the hour hand will have turned less than one full revolution (360 degrees).

As the hour hand turns from its initial position to its final position, it may or may not sweep past the number 12 on the clock's dial.

If it does, then either the initial time is an "A.M." time and the final time a "P.M." time, or vice versa.

If it does not, then either both times (initial and final) are "A.M." or both are "P.M."

Each line of input gives rise to one line of output, containing the initial and final times, and the number of times the minute hand passes the hour hand between the initial time and the final time.

Observe all details of formatting, such as upper/lower case letters, punctuation, blank spaces, and the absence of blank lines.

In each time display, the hours and minutes are displayed in fields of width 2, separated by a colon.

The ten's digit (of hours or minutes) is displayed as a zero if it is zero.

Here is a formatting template shown between two lines of the above Output:

```

Initial time  Final time  Passes
12345678901234567890123456789012
      12:50      01:02      0

```

## Problem Source

Rocky Mountain 2000

## 【题目大意】

给出一只 12 小时的模拟显示钟，有一个时针和一个分针。在给定的时间里，分针经过时针多少次？

输入有很多行，每行 4 个数。

第一对数表示“开始时间”，第二对数表示“结束时间”。

每对数的第一个数表示小时，第二个数表示分钟。

小时的范围 1~12，分钟的范围 0~59。

在开始时间和结束时间，分针不会经过时针。（特别是开始和结束时间里不会有 12 00。）

开始时间和相应的结束时间不相同。

从开始时间到相应的结束时间，时针转过的度数小于  $360^\circ$ 。

当时针从开始时间转到结束时间时，它也许会经过钟表上的 12。

如果时针转过 12 点，那么初始时间是“A.M.”，结束时间是“P.M.”。反之亦然。

如果时针没转过 12 点，那么初始时间与结束时间都是“A.M.”或“P.M.”。

输出对应输入的每一行，包括初始时间，结束时间和在此段时间里时针与分针相遇的次数。

注意输出格式，像大写/小写字母、标点、空格，出现和缺少的空行。

输出的小时和分钟之间用冒号分开，宽度为 2。

如果（小时及分钟的）十位是 0，应该输出十位的 0。

## 【算法分析】

计算在给定时间段内，分针经过时针的次数。

设开始和结束时间：

```
int startHour, startMinute, finalHour, finalMinute;
```

设分针经过的分钟刻度为

```
int sMinute, fMinute;
```

从 00:00 起（因此时钟要模 12），分针经过的分钟刻度为

```
sMinute = (startHour%12)*60 + startMinute;
```

```
fMinute = (finalHour%12)*60 + finalMinute;
```

分针转动 12 小时，经过的分钟刻度为

$$12 \times 60 = 720$$

因为 12 小时内分针与时针最多相遇 11 次（从 00:00 起的 1 小时内，分针与时针不相遇）。因此分针经过时针的次数为

```
pass = (fMinute*11)/720 - (sMinute*11)/720;
```

如果分针的起点大于终点，则 pass 是负数：

```
if (sMinute > fMinute) pass += 11;
```

## 【程序代码】

---

程序名称: zju1122.c

题 目: Clock

提交语言: C

运行时间: 0ms

运行内存: 160KB

---

```
#include<stdio.h>
int startHour[100];
int main()
{
    int startHour,startMinute,finalHour,finalMinute;//开始和结束时间
    int sMinute,fMinute;                          //分针经过的分钟刻度
    int pass;                                       //分针经过时针的次数
    printf("Program 3 by team X\n");
    printf("Initial time  Final time  Passes\n");

    while(scanf("%d%d%d%d",&startHour, &startMinute, &finalHour,
                &finalMinute)!=EOF)
    {
        //计算分针经过的分钟刻度（从 00:00 起）
        sMinute = (startHour%12)*60 + startMinute;
        fMinute = (finalHour%12)*60 + finalMinute;
        //计算经过的次数
        pass = (fMinute*11)/720 - (sMinute*11)/720;
        //分针的起点大于终点
        if (sMinute>fMinute) pass += 11;
        printf("      %02d:%02d", startHour, startMinute);
        printf("      %02d:%02d%8d\n", finalHour, finalMinute, pass);
    }
    printf("End of program 3 by team X\n");
    return 0;
}
```

## ZJU1139-Rectangles<sup>[1、2、3]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768KB

---

Specialist in VLSI design testing must decide if there are some components that cover each

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1139>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1468>

[3] <http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2043>

other for a given design. A component is represented as a rectangle. Assume that each rectangle is rectilinearly oriented (sides parallel to the  $x$  and  $y$  axis), so that the representation of a rectangle consists of its minimum and maximum  $x$  and  $y$  coordinates.

Write a program that counts the rectangles that are entirely covered by another rectangle.

The input contains the text description of several sets of rectangles. The specification of a set consists of the number of rectangles in the set and the list of rectangles given by the minimum and maximum  $x$  and  $y$  coordinates separated by white spaces, in the format:

```
nr_rectangles
xmin1 xmax1 ymin1 ymax1
xmin2 xmax2 ymin2 ymax2
...
xminn xmaxn yminn ymaxn
```

The output should be printed on the standard output. For each given input data set, print one integer number in a single line that gives the result (the number of rectangles that are covered).

## Input

```
3
100 101 100 101
0 3 0 101
20 40 10 400
4
10 20 10 20
10 20 10 20
10 20 10 20
10 20 10 20
```

## Output

```
0
4
```

## Problem Source

Southeastern Europe 2000

### 【题目大意】

在测试超大规模集成电路时，对给定的一个设计，专家要检测元件是否相互覆盖。一个元件当作一个长方形。假设每个长方形都是水平排列（每边与  $x$  和  $y$  轴平行），所以长方形由最小的和最大的  $x$ 、 $y$  坐标表示。

编程计算被完全覆盖的长方形个数。

输入有多组长方形实例。对每组长方形，第一个数字是长方形的数量，然后是长方形的最小和最大  $x$ 、 $y$  坐标（坐标之间有空格）。

对每组输入数据，输出一行，是被完全覆盖的长方形数量。

## 【算法分析】

由于运行时间是 10ms，简单枚举就可以通过。在枚举时注意，当发现一个长方形被另一个长方形完全覆盖时，就不能再继续搜索，这样会重复计数的。

## 【程序代码】

---

程序名称: zju1139.c

题 目: Rectangles

提交语言: C

运行时间: 890ms

运行内存: 316MB

---

```
#include<stdio.h>
#define N 10001

//表示长方形的结构体
struct{
    int minx,miny;
    int maxx,maxy;
}a[N];                                //存放所有长方形的数组

int main()
{
    int i,j;
    int n;                            //长方形的个数
    int sum;                          //被完全覆盖的长方形个数
    while(scanf("%d",&n)!=EOF)
    {
        for(i=0;i<n;i++)
            scanf("%d%d%d%d",&a[i].minx, &a[i].maxx, &a[i].miny,
                &a[i].maxy);
        //枚举法，计算被完全覆盖的长方形个数
        sum=0;
        for(i=0;i<n;i++)
            for(j=0;j<n;j++)
            {
                if(j!=i &&
                    a[i].minx>=a[j].minx &&
                    a[i].maxx<=a[j].maxx &&
                    a[i].miny>=a[j].miny &&
                    a[i].maxy<=a[j].maxy)
                    {sum++;break;}    //注意：break的作用是避免重复计算
            }
        printf("%d\n",sum);
    }
    return 0;
}
```

# ZJU1151-Word Reversal<sup>[1, 2]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

For each list of words, output a line with each word reversed without changing the order of the words.

## This problem contains multiple test cases!

The first line of a multiple input is an integer  $N$ , then a blank line followed by  $N$  input blocks. Each input block is in the format indicated in the problem description. There is a blank line between input blocks.

The output format consists of  $N$  output blocks. There is a blank line between output blocks.

## Input

You will be given a number of test cases. The first line contains a positive integer indicating the number of cases to follow. Each case is given on a line containing a list of words separated by one space, and each word contains only uppercase and lowercase letters.

## Output

For each test case, print the output on one line.

## Sample Input

```
1

3
I am happy today
To be or not to be
I want to win the practice contest
```

## Sample Output

```
I ma yppah yadot
oT eb ro ton ot eb
I tnaw ot niw eht ecitcarp tsetnoc
```

## Problem Source

East Central North America 1999, Practice

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1151>

[2] <http://plg1.cs.uwaterloo.ca/~acm00/regionals/>



## 【题目大意】

对于每行单词，输出相应的一行：单词倒过来输出，但不改变单词的顺序。

本题包含多组测试例！

多组测试例的第一行是一个整数  $N$ ，然后是一个空行，接着是  $N$  个输入数据块。每个数据块的格式在问题描述中给出。每个数据块之间有一个空行。

输出格式包括  $N$  个输出数据块，每个输出数据块之间有一个空行。

### 输入格式

每个数据块有多个测试例。第一行是一个正整数，说明测试例的数量。每个测试例一行，是由一个空格分开的单词，每个单词仅包含大小写字母。

### 输出格式

对每个测试例，输出一行。

## 【算法分析】

每次读取一行单词；确定每个单词的范围，在该范围内，把单词的字符顺序反过来。

## 【程序代码】

---

程序名称：	zju1151.c
题    目：	Word Reversal
提交语言：	C
运行时间：	0ms
运行内存：	160KB

---

```
#include<stdio.h>
#include<string.h>
char line[1000];

int main()
{
    int n;                //数据块的个数
    int m;                //每个数据块中的测试例数
    int len;              //每行单词的长度
    int i,j,k;
    scanf("%d",&n);
    while (n--)
    {
        scanf("%d\n",&m);
        while (m--)
        {
            //读取一行单词
            gets(line);
            len = strlen(line);
            j = 0;
            for(i=0; i<len; i++)
            {
```

```

        //定位一个单词
        j = i+1;
        while(j<len && line[j]!=' ') j++;
        //将该单词的字符顺序反过来
        for(k=0; k<(j-i)/2; k++)
        {
            char c = line[i+k];
            line[i+k] = line[j-1-k];
            line[j-1-k] = c;
        }
        i = j;
    }
    //输出构造之后的一行
    puts(line);
}
if (n) printf("\n");
}
return 0;
}

```

## ZJU1152-A Mathematical Curiosity<sup>[1, 2]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

Given two integers  $n$  and  $m$ , count the number of pairs of integers  $(a,b)$  such that  $0 < a < b < n$  and  $(a^2 + b^2 + m) / (ab)$  is an integer.

### This problem contains multiple test cases!

The first line of a multiple input is an integer  $N$ , then a blank line followed by  $N$  input blocks. Each input block is in the format indicated in the problem description. There is a blank line between input blocks.

The output format consists of  $N$  output blocks. There is a blank line between output blocks.

### Input

You will be given a number of cases in the input. Each case is specified by a line containing the integers  $n$  and  $m$ . The end of input is indicated by a case in which  $n=m=0$ . You may assume that  $0 < n \leq 100$ .

### Output

For each case, print the case number as well as the number of pairs  $(a,b)$  satisfying the given

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1152>

[2] <http://plg1.cs.uwaterloo.ca/~acm00/regionals/>

property. Print the output for each case on one line in the format as shown below.

## Sample Input

```
1
10 1
20 3
30 4
0 0
```

## Sample Output

```
Case 1 : 2
Case 2 : 4
Case 3 : 5
```

## Problem Source:

East Central North America 1999, Practice

### 【题目大意】

给出两个整数  $n$  和  $m$ ，统计整数对  $(a, b)$  的个数，满足  $0 < a < b < n$  并且  $(a^2 + b^2 + m) / (ab)$  是一个整数。

本题包含多组测试例！

多组测试例的第一行是一个整数  $N$ ，然后是一个空行，接着是  $N$  个输入数据块。每个数据块的格式在问题描述中给出。每个数据块之间有一个空行。

输出格式包括  $N$  个输出数据块，每个输出数据块之间有一个空行。

### 输入格式

每个数据块有多个测试例。每个测试例一行，是两个整数  $n$  和  $m$ 。测试例数据是  $n=m=0$  时，输入结束。可以假设  $0 < n \leq 100$ 。

### 输出格式

对每个测试例，输出测试例的编号，满足上述要求的整数对  $(a, b)$  的个数，输出格式如样例输出所示。

### 【算法分析】

由于  $0 < n \leq 100$ ，通过枚举算法就可以找到答案。

注意条件： $0 < a < b < n$ ，枚举时， $b$  从  $a+1$  开始。

### 【程序代码】

---

程序名称：	zju1152.c
题    目：	A Mathematical Curiosity
提交语言：	C
运行时间：	110ms
运行内存：	160KB

---

```

#include <stdio.h>

int main() {
    int i;
    int iCase;                //输入数据块的个数
    int a, b;                 //整数对 (a, b)
    int n, m;                 //输入的两个整数
    int count;                //满足条件的整数对 (a, b) 的个数
    scanf ("%d", &iCase);
    for (i=0; i<iCase; i++) {
        if (i) printf("\n");
        int number = 1;       //测试例编号
        while (scanf("%d%d", &n, &m) && (n||m)) {
            count = 0;
            //枚举算法
            for(a = 1; a < n; a++)
                for(b = a + 1; b < n; b++)
                    if (!(a*a+b*b+m)%(a*b)) count++;
            printf("Case %d: %d\n", number++, count);
        }
    }
}

```

## ZJU1154-Niven Numbers<sup>[1、2]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

A Niven number is a number such that the sum of its digits divides itself. For example, 111 is a Niven number because the sum of its digits is 3, which divides 111. We can also specify a number in another base  $b$ , and a number in base  $b$  is a Niven number if the sum of its digits divides its value.

Given  $b(2 \leq b \leq 10)$  and a number in base  $b$ , determine whether it is a Niven number or not.

### This problem contains multiple test cases!

The first line of a multiple input is an integer  $N$ , then a blank line followed by  $N$  input blocks. Each input block is in the format indicated in the problem description. There is a blank line between input blocks.

The output format consists of  $N$  output blocks. There is a blank line between output blocks.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1154>

[2] <http://plg1.cs.uwaterloo.ca/~acm00/regionals/>

## Input

You will be given a number of test cases. Each line of input contains the base  $b$ , followed by a string of digits representing a positive integer in that base. There are no leading zeroes. The input is terminated by a line consisting of 0 alone.

## Output

For each case, print "yes" on a line if the given number is a Niven number, and "no" otherwise.

## Sample Input

```
1
10 111
2 110
10 123
6 1000
8 2314
0
```

## Sample Output

```
yes
yes
no
yes
no
```

## Problem Source

East Central North America 1999, Practice

### 【题目大意】

一个 Niven 数就是一个它的各位数字之和能整除它本身的数。例如：111 是 Niven 数，因为各位数字之和是 3，能够整除 111。同样地，对其他数制  $b$ ，在该进制下，如果一个数的各位数字之和能够整除该数字，也是一个 Niven 数。

给出数制  $b$  ( $2 \leq b \leq 10$ ) 和一个  $b$  数制的数字，判断这个数是否是 Niven 数。

本题包含多组测试例！

多组测试例的第一行是一个整数  $N$ ，然后是一个空行，接着是  $N$  个输入数据块。每个数据块的格式在问题描述中给出。每个数据块之间有一个空行。

输出格式包括  $N$  个输出数据块，每个输出数据块之间有一个空行。

### 输入格式

第一行是测试例数。每个测试数据占一行，先是数制  $b$ ，之后是一个数字串，表示该数制中的一个数。一行只有一个 0 时，表示该数据块输入结束。

### 输出格式

对每个测试例，如果该数是一个 Niven 数，就输出 “yes”，否则输出 “no”。

## 【算法分析】

关于 Niven 数的更多描述，请参考网站：[http://en.wikipedia.org/wiki/Harshad\\_number](http://en.wikipedia.org/wiki/Harshad_number)，[http://wapedia.mobi/en/Niven\\_number](http://wapedia.mobi/en/Niven_number)，中文 <http://zh.wikipedia.org/wiki> 中，搜索“哈沙德数”。

对任意进制  $b$ ，都转化为十进制数进行计算，然后按要求进行判断，这样计算就变得十分简单。

## 【程序代码】

---

程序名称:	zju1154.c
题    目:	Niven Numbers
提交语言:	C
运行时间:	0ms
运行内存:	160KB

---

```
#include <stdio.h>
#include <string.h>

int main()
{
    int n;                //数据块的数量
    int b;                //进制
    char a[100];          //一个数字串
    int i;

    scanf("%d", &n);
    while(n--)
    {
        while(scanf("%d", &b) && b)
        {
            scanf("%s", a);
            int len = strlen(a);
            int iSum = 0;                //各位数字之和
            int iNum = 0;                //在十进制下的值
            for(i = 0; i < len; i++)
            {
                iSum += a[i] - '0';
                iNum = iNum * b + a[i] - '0';    //a[i]是字符数字
            }
            if(iNum % iSum == 0) printf("yes\n"); //是一个Niven数
            else printf("no\n");
        }
        if(n) printf("\n");
    }
    return 0;
}
```

## 第二章 模拟算法题

在本章的题目主要是模拟算法题。根据题目的要求逐步模拟，实现题目的要求。

### ZJU1088-System Overload<sup>[1、2、3]</sup>

---

Time limit: 10 Seconds    Memory limit: 32768K

---

Recently you must have experienced that when too many people use the BBS simultaneously, the net becomes very, very slow.

To put an end to this problem, the Sysop has developed a contingency scheme for times of peak load to cut off net access for some buildings of the university in a systematic, totally fair manner. Our university buildings were enumerated randomly from 1 to  $n$ . XWB is number 1, CaoGuangBiao (CGB) Building is number 2, and so on in a purely random order.

Then a number  $m$  would be picked at random, and BBS access would first be cut off in building 1 (clearly the fairest starting point) and then in every  $m$ th building after that, wrapping around to 1 after  $n$ , and ignoring buildings already cut off. For example, if  $n=17$  and  $m=5$ , net access would be cut off to the buildings in the order [1,6,11,16,5,12,2,9,17,10,4,15,14,3,8,13,7]. The problem is that it is clearly fairest to cut off CGB Building last (after all, this is where the best programmers come from), so for a given  $n$ , the random number  $m$  needs to be carefully chosen so that building 2 is the last building selected.

Your job is to write a program that will read in a number of buildings  $n$  and then determine the smallest integer  $m$  that will ensure that our CGB Building can surf the net while the rest of the university is cut off.

### Input Specification

The input file will contain one or more lines, each line containing one integer  $n$  with  $3 \leq n < 150$ , representing the number of buildings in the university.

Input is terminated by a value of zero (0) for  $n$ .

### Output Specification

For each line of the input, print one line containing the integer  $m$  fulfilling the requirement

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1088>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2244>

[3] <http://acm.uva.es/p/v4/440.html>

specified above.

### Sample Input

3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
0

### Sample Output

2  
5  
2  
4  
3  
11  
2  
3  
8  
16

### 【题目大意】

最近，当很多人同时使用 BBS 时，你肯定经历过网速变得很慢很慢这种情况。

为了解决这个问题，在下载高峰时段，系统管理员制定了一个非常公平的应急计划：切断大学里一些大楼的网络访问。我们大学里的大楼被任意地编号为  $1 \sim n$ 。XWB 是 1 号，CaoGuangBiao (CGB) 楼是 2 号，等等，纯粹都是任意的顺序。

接着挑选一个随机数  $m$ ，首先切断 1 号大楼（显然，这是最公平的起始位置），然后从 1 到  $n$  循环，每隔  $m$  个大楼就切断其网络连接，但是不能把已经切断网络连接的大楼重新计算在内。例如，如果  $n=17$ ， $m=5$ ，切断网络连接的大楼依次是[1, 6, 11, 16, 5, 12, 2, 9, 17, 10, 4, 15, 14, 3, 8, 13, 7]。问题是如何以最公平的方式，确保 CGB 大楼在最后切断网络连接（毕竟，这是盛产优秀程序员的地方）。因此对给定的数  $n$ ，我们得选好随机数  $m$ ，确保 2 号大楼被最后选到。

**编程任务：**对现有的大楼总数  $n$ ，确定最小整数  $m$ ，当大学中其他大楼的网络都已切断时，以便我们的 CGB 大楼仍能网上冲浪。

### 输入格式

输入有一行或多行，每行包含一个整数  $n$  ( $3 \leq n < 150$ )，表示大学中大楼的数目。

当  $n$  为零 (0) 时，输入结束。

### 输出格式

对于每行输入，输出一行  $m$ ， $m$  应满足上述要求。

### 【算法分析】

本题理解起来比较简单，相当于报数问题，即约瑟夫 (joseph) 环问题。解决该问题的最简单办法是模拟，由于  $n < 150$ ，时限是 10 秒，模拟应该不会超时。但是要获得快速的解决算法，还是比较困难的，请参考本题的【其他算法】。

可以采用链表或者数组的方法模拟报数过程，本题使用数组。设变量  $n$  表示大楼总数，变量  $m$  表示切断网络连接的大楼间隔，将大楼  $1 \sim n$  的编号保存到数组 `next` 中：

```
#define MaxN 150
int next[MaxN];
```

切断网络连接的大楼编号保存到数组 `cut` 中：



```
bool cut[MaxN];
```

令  $m$  从 1 开始模拟，每次增加 1。当从 1 号大楼开始切断网络连接，直到最后只剩下 2 号大楼时的  $m$ ，便是所求的数。关于模拟过程的详细描述，在一般的程序设计教材中都有，这里不再赘述。

## 【程序代码】

---

程序名称:     zju1088.c

题      目:     System Overload

提交语言:     C++

运行时间:     00:00.68

运行内存:     396KB

---

```
#include<stdio.h>
#include<memory.h>

#define MaxN 150

int n;                                //大楼总数
int m;                                //切断网络连接的大楼间隔
int next[MaxN];                       //保存大楼 1~n 的编号
bool cut[MaxN];                       //切断网络连接的大楼编号

int main()
{
    while(scanf("%d", &n) && n)
    {
        int i;
        //保存大楼 1~n 的编号，注意保存的顺序
        next[n] = 1;
        for(i = 1; i <= n - 1; i++)
            next[i] = i + 1;
        m = 1;                        //从 1 开始模拟
        bool find = false;            //找到满足条件 m 的标志
        while(!find)
        {
            m++;
            memset(cut, true, sizeof(cut));
            int from = 1;              //首先切断 1 号大楼
            cut[1] = false;
            int number = n - 1;        //剩下的大楼数量
            int nextCut, jump;
            while(number > 1)          //剩下不止 1 幢大楼时
            {
                //查找下一个要切断的大楼
                nextCut = from;
                jump = 0;
                while(jump < m)
```

```

    {
        nextCut = next[nextCut];
        if (cut[nextCut]) jump++;
    }
    cut[nextCut] = false;        //切断该大楼
    number--;                    //大楼的数量少 1 个
    from = nextCut;
    //2 号大楼不是最后一个切断，需要重新计算 m
    if (from == 2 && number != 1) break;
}
//如果最后一次切断的是 2 号大楼，结束查找
if (number == 1 && cut[2] == true)
    find = true;
}
printf("%d\n", m);
}
return 0;
}

```

## 【其他算法】

无论是用链表还是用数组实现约瑟夫环问题，都有一个共同点：要模拟整个游戏过程，时间复杂度都是  $O(nm)$ ，当  $n, m$  非常大的时候，相当费时。这里提供一个快速的算法，读者可以参考网友winsty博客<sup>[1]</sup>中详细的算法说明。

## 【程序代码】

---

程序名称：	zju1088-math.c
题    目：	System Overload
提交语言：	C
运行时间：	00:00.03
运行内存：	392K

---

```

#include <stdio.h>

int m, n;
//检查 m 是否满足条件
int check() {
    int i;
    //使用递推公式计算
    int from = 0;
    for (i = 2; i < n; ++i)
        from = (from + m) % i;
    if (from == 0) return 1;
    else return 0;
}

```

---

[1] <http://acm.zjuwinsty.cn/post/25.html>

```

int main(){
    while (scanf("%d", &n) && n){
        //m 从 1 开始起步测试
        m = 1;
        while (!check()) ++m;
        printf("%d\n", m);
    }
    return 0;
}

```

## ZJU1098-Simple Computers<sup>[1, 2]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768K

---

You are to write an interpreter for a simple computer. This computer uses a processor with a small number of machine instructions. Furthermore, it is equipped with 32 byte of memory, one 8-bit accumulator (accu) and a 5-bit program counter (pc). The memory contains data as well as code, which is the usual von Neumann architecture.

The program counter holds the address of the instruction to be executed next. Each instruction has a length of 1 byte-the highest 3 bits define the type of instruction and the lowest 5 bits define an optional operand which is always a memory address (xxxxx). For instructions that don't need an operand the lowest 5 bits have no meaning ( - - - - - ). Here is a list of the machine instructions and their semantics:

000x	x x x x	STA x	store the value of the accu into memory byte <i>x</i>
001x	x x x x	LDA x	load the value of memory byte <i>x</i> into the accu
010x	x x x x	BEQ x	if the value of the accu is 0 load the value <i>x</i> into the pc
011-	- - - -	NOP	no operation
100-	- - - -	DEC	subtract 1 from the accu
101-	- - - -	INC	add 1 to the accu
110x	x x x x	JMP x	load the value <i>x</i> into the pc
111-	- - - -	HLT	terminate program

In the beginning, program counter and accumulator are set to 0. After fetching an instruction but before its execution, the program counter is incremented. You can assume that programs will terminate.

## Input Specification

The input file contains several test cases. Each test case specifies the contents of the memory

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1098>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2410>

prior to execution of the program. Byte 0 through 31 are given on separate lines in binary representation. A byte is denoted by its highest—to—lowest bits. Input is terminated by EOF.

## Output Specification

For each test case, output on a line the value of the accumulator on termination in binary representation, again highest bits first.

## Sample Input

00111110	00000000	00111111	00000000
10100000	00000000	10000000	00000000
01010000	00000000	00000010	00000000
11100000	00000000	11000010	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	11111111
00000000	00000000	00000000	10001001

## Sample Output

10000111

## Problem Source

University of Ulm Local Contest 2000

### 【题目大意】

你为一台简单的计算机编写解释程序。该计算机的处理器使用很少的机器指令。此外，它的存储器是 32 字节、一个 8 位的累加器（accu）和一个 5 位的程序计数器（pc）。存储器既存储数据也存储代码，这就是通常的冯·诺伊曼（von Neumann）结构。

程序计数器里保存下一条运行指令的地址。每条指令占 1 字节，其中高 3 位代表指令类型，低 5 位代表可选的操作数，通常是存储器位址（xxxxx）。如果指令不需要操作数时，其低 5 位就没有任何意义（-----）。下面是机器指令和语法：

000x x x x x	STA $x$	把 accu 的值存放到 memory 的第 $x$ 字节
001x x x x x	LDA $x$	把 memory 的第 $x$ 字节值传送到 accu 中
010x x x x x	BEQ $x$	如果 accu 的值是 0，把 $x$ 的值传送到 pc 中
011- - - - -	NOP	无操作
100- - - - -	DEC	accu 的值减 1
101- - - - -	INC	accu 的值加 1
110x x x x x	JMP $x$	把 $x$ 的值传送到 pc
111- - - - -	HLT	程序结束

开始时，程序计数器和累加器置 0。取出一条指令之后，但该指令还没有运行，程序计数器的值就增加。可以假定，程序会终止的。

### 输入格式:

输入包含多组测试例。每组测试例描述程序运行之前的存储器内容。字节 0~31 (用二进制表示), 每行一个字节。一个字节由高/低位表示。遇到 EOF 时, 输入结束。

### 输出格式:

对每个测试例, 输出一行, 是程序结束时累加器的值 (用二进制表示), 高位在前面。

### 【算法分析】

这是一道模拟题, 处理读取的数据有点麻烦。

把 32 条指令由字符型数据转换成数字型数据, 就可以使用二进制的位运算, 编程就方便很多了。

#### (1) 数据结构

首先使用一个过渡数组, 把字符型指令读出来:

```
char x[32][9];
```

然后把字符型数据转换成数字型数据, 保存到要处理的数组中:

```
unsigned char memory[32];
```

由于使用了两个不同的数组, 转换过程就比较简单。

#### (2) 模拟指令的运行

指令的获取: 将每条指令右移 5 位, 得到了高 3 位, 这就是指令的编码。然后按每条指令的要求操作就可以了。

### 【程序代码】

---

程序名称:	zju1098.cpp
题 目:	Simple Computers
提交语言:	C++
运行时间:	00:00.00
运行内存:	260KB

---

```
#include <stdio.h>
#include <memory.h>
//用枚举型数据对指令进行编码
enum INST{STA, LDA, BEQ, NOP, DEC, INC, JMP, HLT};

unsigned char memory[32];           //存放数字型的指令
unsigned char accu;                 //累加器
unsigned char PC;                   //程序计数器

//模拟指令的运行
void exec()
{
    while(true)
    {
        switch(memory[PC] >> 5)    //获取指令的编码
        {
            //模拟 STA 指令的运行
            case STA:
                memory[memory[PC] & 0x1F] = accu;
```

```

        PC = ++PC & 0x1F;
        break;
//模拟 LDA 指令的运行
case LDA:
    accu = memory[memory[PC] & 0x1F];
    PC = ++PC & 0x1F;
    break;
//模拟 BEQ 指令的运行
case BEQ:
    if(accu == 0) PC = memory[PC] & 0x1F;
    else PC = ++PC & 0x1F;
    break;
//模拟 NOP 指令的运行
case NOP:
    PC = ++PC & 0x1F;
    break;
//模拟 DEC 指令的运行
case DEC:
    accu--;
    PC = ++PC & 0x1F;
    break;
//模拟 INC 指令的运行
case INC:
    accu++;
    PC = ++PC & 0x1F;
    break;
//模拟 JMP 指令的运行
case JMP:
    PC = memory[PC] & 0x1F;
    break;
//模拟 HLT 指令的运行
case HLT:
    //指令运行完毕时，要输入累加器中的值
    //从高位到低位，按位输出（移位与 1 运算）
    for(int i=7; i>=0; i--)
        printf("%d", (accu>>i)&1);
    printf("\n");
    return;
}
}

int main()
{
    int i, j;
    //存放字符型的指令

```

```

char x[32][9];
//读取第一条指令, 并判断结束符
while (scanf("%s", x[0]) != EOF)
{
    //读取其余 31 条指令
    for(i = 1; i < 32; i++)
        scanf("%s", x[i]);
    memset(memory, 0, sizeof(memory));
    //把字符型数据转换成数字型数据
    for(i = 0; i < 32; i++)
        for(j = 0; j < 8; j++)
            memory[i] = (memory[i] << 1) | (x[i][j] - '0');
    PC = 0x0;
    accu = 0x0;
    //调用模拟指令运行的程序
    exec();
}
return 0;
}

```

## ZJU1121-Reserve Bookshelf <sup>[1、2、3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

A very small rural library has only one shelf dedicated to reserve books. (Reserve books are deemed important to a large number of library users and can therefore be checked out only for very short periods of time.)

- When the library's staff places a book on reserve, the book is put on the reserve shelf's left end. All books that were already on the reserve shelf must therefore be moved to the right, just enough to make room for the new book on the shelf.

- When a reserve book is checked out and subsequently returned, it is also put on the reserve shelf's left end, as if it were just being placed on reserve.

- What happens when the shelf is full? When a book is placed on reserve, or when a previously checked out reserve book is returned, and there is not enough room on the reserve bookshelf, then, starting at the right end, as many books (but no more) are removed from the reserve shelf (they are "taken off reserve") as needed in order to make room for the book to be put on the reserve bookshelf.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1121>

[2] <http://acm.uva.es/archive/nuevportal/data/problem.php?p=2076>

[3] <http://acm.tju.edu.cn/toj/showp2094.html>

## Sample Input

```
250
PRINT
ADD      101 Uses for a Dead Cat      38
ADD      Thin Thighs in 30 Days      63
ADD      The Republic                44
ADD      Mein Kampf                  101
PRINT
ADD      Principia Mathematica       79
PRINT
CHECKOUT The Republic
ADD      On the Origin of Species    55
ADD      The C Programming Language  15
PRINT
RETURN   The Republic
ADD      101 Uses for a Dead Cat      38
CHECKOUT The C Programming Language
PRINT
```

## Sample Output

```
Program 2 by team X
AVAILABLE SHELF SPACE:      250

Mein Kampf                  101
The Republic                44
Thin Thighs in 30 Days      63
101 Uses for a Dead Cat     38
AVAILABLE SHELF SPACE:      4

Principia Mathematica       79
Mein Kampf                  101
The Republic                44
AVAILABLE SHELF SPACE:      26

The C Programming Language   15
On the Origin of Species     55
Principia Mathematica       79
Mein Kampf                  101
AVAILABLE SHELF SPACE:      0

101 Uses for a Dead Cat     38
The Republic                44
On the Origin of Species     55
Principia Mathematica       79
AVAILABLE SHELF SPACE:      34
```



End of program 2 by team X

The first line of the input file contains a single integer, the total available space on the reserve book shelf (when it is empty), measured in millimeters (1 inch equals about 25 millimeters). Its value will be at least 250 and at most 1500.

Your program will start with an empty reserve book shelf.

Each subsequent line of input will start with one of the four commands ADD, CHECKOUT, RETURN, PRINT in upper case letters, starting in column 1.

There will not be any trailing blank spaces in the input file.

- The ADD command:

- The command is followed by a title (starting in column 10), consisting of at most 29 printable characters.

- The title is followed (starting in column 40) by the thickness of the book (the amount of book shelf space it will require), a positive integer, not greater than 150.

- The effect of the ADD command will be to place a new title on reserve, as described above.

- Duplicate titles:

Once a particular title has been ADDED, it will not be ADDED again as long as that book is on the reserve book shelf or has been checked out. However, if a title has been forced off the reserve book shelf (by another ADD command or by a RETURN command) then that title may appear in a subsequent ADD command (with the same thickness as before).

- The CHECKOUT command:

- The command is followed by a title (starting in column 10) which is currently on the reserve shelf.

- The effect of the CHECKOUT command is to remove the specified title from the reserve shelf (until it is placed back on the reserve shelf by a subsequent RETURN command).

- A CHECKOUT may create a gap between two books (for example, when "The C Programming Language" is checked out in the example shown). Such a gap will count as part of the free space.

- The RETURN command:

- The command is followed by a title (starting in column 10) which is currently checked out.

- The effect of the RETURN command is to put the specified book to the left end of the reserve shelf, as described in the introductory paragraphs.

- The PRINT command:

- The effect of the command will be to display in the Output:

- the title and thickness of each book currently on the reserve shelf, one book per line, in the (left—to—right) order in which the books are currently on the reserve shelf,

- the currently available space on the reserve shelf, and

- one blank line.

Pay attention to all formatting details, such as punctuation, upper/lower case characters, and the

presence or absence of blank spaces and blank lines.

All titles, as well as the sentence "AVAILABLE SHELF SPACE:" are left-justified, starting in column 1.

All numerical quantities are displayed right-justified, with the unit's digit in column 34.

## Problem Source:

Rocky Mountain 2000

### 【题目大意】

一个非常小的乡村图书馆只有一个书架存放书籍。（对于有大量的图书馆读者来说，图书存放是非常重要的，并且登记借书只需要很少的时间。）

图书管理员始终从书架的左端存放书籍。为了给新书留出足够的空间，书架上所有的图书将往右边移。

当借出的图书归还的时候，也像原先一样从书架左端开始放起。

书架放满了怎么办呢？当新书或还来的书放不进书架时，图书管理员就从书架右边撤下同样数量的书（即“下架”），以便把新书放进去。

.....

输入数据的第一行是一个整数，书架上的空间长度（毫米，1 英寸大约 25 毫米）。该值范围是 250~1500。

程序开始时，书架是空的。

随后的每行是命令行，开头分别是 ADD, CHECKOUT, RETURN, PRINT。命令全部是大写字母，从第一列开始。

每行后面没有空格。

#### ● ADD 命令：

➤ 命令后面是书籍名（从第 10 列开始），至多 29 个字符。

➤ 书籍后面是该书的厚度（从第 40 列开始，该书占用的空间位置）。厚度是一个正整数，不大于 150。

➤ ADD 命令作用是增加一本新书。

➤ 重名：当一本书存放后，不管该书是在书架上还是已经借出，重名的书就不能再放上去了。但是，假如某本书由于 ADD 命令或 RETURN 命令而被管理员从书架上挪去，那么它还会出现在之后的 ADD 命令里（厚度自然没变）。

#### ● CHECKOUT 命令：

➤ 命令后是书籍名（从第 10 列开始，且已经在书架上了）。

➤ CHECKOUT 命令的作用是把书从书架上拿走（直到 RETURN 命令重新把书放回到书架上）。

➤ CHECKOUT 命令会造成书与书之间的空隙。（例如，示例中“The C Programming Language”被借出）。空隙也是书架剩余空间的一部分。

#### ● RETURN 命令：

➤ RETURN 命令后是书籍名（从第 10 列输入）。该书已被借出。

➤ RETURN 命令的作用是将书放到书架的左端，如前面的描述。

## ● PRINT 命令:

➤ PRINT 命令的作用是输出。

➤ 以书架上从左往右的顺序, 输出书架上每本书的名称和厚度, 每本书一行。

➤ 书架上的有用空间。

➤ 一个空行。

注意格式, 像标点、大写/小写字符、出现或缺少的空格和空行。

所有的书名和句子“AVAILABLE SHELF SPACE:”都是左对齐的, 从第一列开始。

数字都是右对齐的。从第 34 列开始。

## 【算法分析】

本题是处理字符串的, 处理的方法是按题目要求进行模拟的, 实现 ADD, CHECKOUT, RETURN, PRINT 命令的功能。

### (1) 数据结构

书名是字符串, 采用字符串数组的办法, 查找起来比较麻烦, 这里采用 C++标准模板库函数 map()容器:

```
map<string,int> books;
```

其中“key”是书名, “value”是书的编号。样例的书名和编号如表 2-1 所示。

表 2-1 样例的书名和编号

key	value
101 Uses for a Dead Cat	8
Mein Kampf	3
On the Origin of Species	5
Principia Mathematica	4
The C Programming Language	6
The Republic	2
Thin Thighs in 30 Days	1

注意: 书名“101 Uses for a Dead Cat”在第二次 ADD 时, 编号由 0 改为 8。

每本书的参数由下列变量表示:

```
bool shelf[50];           //是否在架
char title[50][40];       //书名
int serial[50];           //书的编号
```

书的参数应该用结构体数组表示, 但那样的话, 代码很长。

书架空间用下列变量表示:

```
int placed;               //已经占用的空间
int space;                //书架总空间
```

### (2) 模拟四条命令

四条命令分别用 4 个函数实现的, 其中书从书架下架的代码在 ADD()和 RETURN()中都要用到, 就编写了函数 remove()。

## 【程序代码】

---

程序名称:

题    目:    Reserve Bookshelf

提交语言:    C++

运行时间:    0ms

运行内存:    188KB

---

```
#include<iostream>
#include<map>
using namespace std;

bool shelf[50];                //书是否在架
char title[50][40];           //书名
int serial[50];               //书的编号
map<string,int> books;         //书名及其编号
int number;                   //总的书号
int placed;                   //书架上的书已经占用的空间
int space;                    //书架总空间
char str[40];

//书从书架下架
void remove() {
    int k = 0;                //编号小的书在书架的右边
    //只有当书架上的书已经占用的空间大于书架总空间时才下架
    while(placed>space)
    {
        if(shelf[k])          //该书在架
        {
            shelf[k] = false;
            placed -= serial[k];
        }
        k++;
    }
}

//模拟 PRINT 命令
void PRINT()
{
    int i;
    //输出在架的每一本书, 注意编号是降序的
    for(i=number-1; i>=0; i--)
        if(shelf[i]) printf("%s%4d\n", title[i], serial[i]);
    printf("AVAILABLE SHELF SPACE:      %4d\n\n", space-placed);
    gets(str);                //读完 PRINT 命令后面的部分
}
```

```

//模拟 ADD 命令
void ADD()
{
    int k;
    //读取 ADD 命令后面的空格
    char ch;
    for(k=0; k<6; k++) scanf("%c",&ch);
    gets(title[number]); //读取书名和厚度
    int thick; //书的厚度
    sscanf(title[number]+30, "%d", &thick);
    title[number][30] = 0;
    //获得新书的相关参数
    placed += thick;
    serial[number] = thick;
    books[title[number]] = number;
    shelf[number] = true;
    number++;
    //多余的书从书架下架
    remove();
}

//模拟 CHECKOUT 命令
void CHECKOUT()
{
    int k,i;
    //读取 CHECKOUT 命令后面的空格
    char ch;
    scanf("%c",&ch);
    gets(str); //读取书名和厚度
    for(i=strlen(str); i<30; i++) str[i]=' ';
    str[30] = 0;
    //处理相关借出的参数
    k = books[string(str)]; //得到该书的编号
    shelf[k] = false;
    placed -= serial[k];
}

//模拟 RETURN 命令
void RETURN()
{
    int k,i;
    //读取 RETURN 命令后面的空格
    char ch;
    for(i=0; i<3; i++)
        scanf("%c",&ch);

```

```

    gets(str);                                     //读取书名
    for(i=strlen(str); i<30; i++) str[i]=' ';
    str[30] = 0;
    k = books[string(str)];                         //得到该书的编号
    //除数组 books 外，相当于增加了一本书
    for(i=0; i<31; i++)
        title[number][i] = title[k][i];
    shelf[number] = true;
    serial[number] = serial[k];
    placed += serial[number];
    number++;
    //多余的书从书架下架
    remove();
}

int main()
{
    cout<<"Program 2 by team X"<<endl;
    //读取书架的总空间
    scanf("%d",&space);
    //变量初值
    placed = 0;
    number = 0;
    books.clear();
    memset(shelf, 0, sizeof(shelf));
    string command;
    //读取所有的指令
    while(cin>>command)
    {
        if (command=="PRINT") PRINT();
        else if(command=="ADD") ADD();
        else if(command=="CHECKOUT") CHECKOUT();
        else if(command=="RETURN") RETURN();
    }
    cout<<"End of program 2 by team X"<<endl;
    return 0;
}

```

# ZJU1143-Date Bugs<sup>[1, 2, 3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

There are rumors that there are a lot of computers having a problem with the year 2000. As they use only two digits to represent the year, the date will suddenly turn from 1999 to 1900. In fact, there are also many other, similar problems. On some systems, a 32-bit integer is used to store the number of seconds that have elapsed since a certain fixed date. In this way, when  $2^{32}$  seconds (about 136 Years) have elapsed, the date will jump back to whatever the fixed date is.

Now, what can you do about all that mess? Imagine you have two computers  $C_1$  and  $C_2$  with two different bugs: One with the ordinary Y2K-Bug (i.e. switching to  $a_1=1900$  instead of  $b_1=2000$ ) and one switching to  $a_2=1904$  instead of  $b_2=2040$ . Imagine that the  $C_1$  displays the year  $y_1=1941$  and  $C_2$  the year  $y_2=2005$ . Then you know the following (assuming that there are no other bugs): the real year can't be 1941, since, then, both computers would show the (same) right date. If the year would be 2005,  $y_1$  would be 1905, so this is impossible, too. Looking only at  $C_1$ , we know that the real year is one of the following: 1941, 2041, 2141, etc. We now can calculate what  $C_2$  would display in these years: 1941, 1905, 2005, etc. So in fact, it is possible that the actual year is 2141.

To calculate all this manually is a lot of work. (And you don't really want to do it each time you forgot the actual year.) So, your task is to write a program which does the calculation for you: find the first possible real year, knowing what some other computers say ( $y_i$ ) and knowing their bugs (switching to  $a_i$  instead of  $b_i$ ). Note that the year  $a_i$  is definitely not after the year the computer was built. Since the actual year can't be before the year the computers were built, the year your program is looking for can't be before any  $a_i$ .

## Input

The input contains several test cases, in which the actual year has to be calculated. The description of each case starts with a line containing an integer  $n(1 \leq n \leq 20)$ , the number of computers. Then, there is one line containing three integers  $y_i, a_i, b_i$  for each computer ( $0 \leq a_i \leq y_i < b_i < 10000$ ).  $y_i$  is the year the computer displays,  $b_i$  is the year in which the bug happens (i.e. the first year which can't be displayed by this computer) and  $a_i$  is the year that the computer displays instead of  $b_i$ .

The input is terminated by a test case with  $n=0$ . It should not be processed.

## Output

For each test case, output the line "Case #k: ", where  $k$  is the number of the situation. Then,

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1143>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1044>

[3] <http://acm.uva.es/p/v7/700.html>

output the line “The actual year is  $z$ .”, where  $z$  is the smallest possible year (satisfying all computers and being greater or equal to  $u=\max(a_i)$ ). If there is no such year less than 10000, output “Unknown bugs detected.”.

Output a blank line after each case.

## Sample Input

```
2
1941 1900 2000
2005 1904 2040
2
1998 1900 2000
1999 1900 2000
0
```

## Sample Output

```
Case #1:
The actual year is 2141.
Case #2:
Unknown bugs detected.
```

## Problem Source

Mid—Central European Regional Contest 1999

### 【题目大意】

据说大量的计算机会出现 2000 年问题。因为计算机只用两位数字表示年份，所以日期会从 1999 突然变成 1900。事实上，也有许多其他类似的问题。某些系统用一个 32 位的整数存储从某个特定日期以后的秒数。这样，无论特定日期是什么，当超过  $2^{32}$  秒（大约 136 年）后，日期就会返回到那个特定的日期。

那该怎样处理这些烦人的问题？假设你有两台计算机  $C_1$  和  $C_2$ ，它们的日期错误不同：一台是普通的 Y2K—Bug（也就是  $b_1=2000$  会转换到  $a_1=1900$ ），另一台在  $b_2=2040$  时会转换到  $a_2=1904$ 。假设  $C_1$  显示年份是  $y_1=1941$ ， $C_2$  显示年份  $y_2=2005$ 。那么你可以推出以下结论（假设没有其他系统错误）：真实年份显然不可能是 1941，因为如果是 1941 年的话，两台计算机应该要显示一样的年份。如果真实年份是 2005， $y_1$  就是 1905，这也是不可能的。如果只观察  $C_1$ ，我们知道真实年份应该是下列其中之一：1941，2041，2141，等等。这些年份如果在  $C_2$  上显示，应该会是：1941，1905，2005，等等。所以，事实上，实际的年份为 2141。

手算相当繁琐。（而且如果忘记了真实年份，你也不想每次都去计算。）因此，你的任务就是编写一个程序实现计算：找出第一个可能的真实年份，了解某些计算机显示的年份 ( $y_i$ ) 和日期错误（例如把  $b_i$  年显示成  $a_i$  年）。注意：年份  $a_i$  肯定不是在发明计算机之后的年份。由于在计算机发明以后才有这样的问题出现，所以你要找的实际年份应该在所有的  $a_i$  之后。

### 输入格式

输入有多组测试数据，计算出真实的年份。每组数据的第一行是一个整数  $n$  ( $1 \leq n \leq 20$ )，表示计算机的数量。然后每行是一台计算机的三个整数  $y_i$ ， $a_i$ ， $b_i$  ( $0 \leq a_i \leq y_i < b_i < 10000$ )。  $y_i$  是



计算机显示的年份,  $b_i$  是日期出错的那年 (也就是不能正确显示的第一年),  $a_i$  是计算机显示的错误年份。

输入  $n=0$  结束, 不需要处理。

### 输出格式

对于每组测试数据, 输出一行 “Case #k:”,  $k$  是数据组的序号。然后, 输出一行 “The actual year is  $z$ .”,  $z$  是最小的可能真实年份 (满足所有的计算机, 并且大于等于 “ $u=\max(a_i)$ ”)。如果 10000 以内没有这样的真实年份, 输出 “Unknown bugs detected.”。

每组测试数据之后有一个空行。

### 【算法分析】

采用模拟算法。

#### (1) 数据结构

设  $y$  表示计算机显示的年份,  $a$  表示计算机显示的错误年份,  $dif$  表示日期出错的年份与计算机显示的错误年份的差, 使用结构体数组:

```
struct {
    int y, a;
    int dif;
} a[25];
```

比所有的  $a_i$  都大的年份:

```
int maxy;
```

#### (2) 计算出真实的年份

在 10000 年以内, 从  $maxy$  开始, 逐一搜索满足条件的年份。

设计算机  $i$ , 当某一年  $y$  与计算机显示的年份  $a[i].y$  之差恰好是  $a[i].dif$  的倍数时:

```
(y-a[i].y)%a[i].dif==0
```

$y$  就是该计算机的真实年份。如果  $y$  也是所有计算机的真实年份, 则就是所要计算的真实年份。由于是从  $maxy$  向上计算的, 所以此时的  $y$  必然是最小值。

### 【程序代码】

---

程序名称:     zju1143.c

题      目:     Date Bugs

提交语言:     C

运行时间:     0ms

运行内存:     160KB

---

```
#include<stdio.h>
```

```
struct {
    int y, a;
    int dif;
} a[25];
```

```
//计算机显示的年份, 计算机显示的错误年份
```

```
//日期出错的年份与计算机显示的错误年份的差
```

```
int main()
```

```
{
```

```
    int iCase = 1;
```

```
    //测试例数
```

```

int maxy;                                //最大的  $a_i$ 
int n;                                  //计算机的数量
while(scanf("%d", &n) && n)
{
    int i;
    int b;                                //日期出错的年份
    //读取原始数据, 计算最大的  $a_i$ 
    maxy = 0;
    for(i=0; i<n; i++)
    {
        scanf("%d%d%d", &a[i].y, &a[i].a, &b);
        a[i].dif = b-a[i].a;
        if(a[i].a>maxy) maxy = a[i].a;
    }
    //从 maxy 开始, 逐一搜索满足条件的年份
    int y;
    for(y=maxy; y<10000; y++)
    {
        //对每台计算机, 判断是否都在出错的年份
        for(i=0; i<n; i++)
            if ((y-a[i].y)%a[i].dif!=0) break;
        if (i==n) break;                //找到了
    }
    printf("Case #d:\n", iCase++);
    if (y<10000) printf("The actual year is %d.\n\n", y);
    else printf("Unknown bugs detected.\n\n");
}
return 0;
}

```

## ZJU1144-Robbery<sup>[1、2、3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768 KB

---

Inspector Robstop is very angry. Last night, a bank has been robbed and the robber has not been caught. And this happened already for the third time this year, even though he did everything in his power to stop the robber: as quickly as possible, all roads leading out of the city were blocked, making it impossible for the robber to escape. Then, the inspector asked all the people in the city to watch out for the robber, but the only messages he got were of the form "We don't see him."

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1144>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1104>

[3] <http://acm.uva.es/problemset/v7/707.html>

But this time, he has had enough! Inspector Robstop decides to analyze how the robber could have escaped. To do that, he asks you to write a program which takes all the information the inspector could get about the robber in order to find out where the robber has been at which time.

Coincidentally, the city in which the bank was robbed has a rectangular shape. The roads leaving the city are blocked for a certain period of time  $t$ , and during that time, several observations of the form "The robber isn't in the rectangle  $R_i$  at time  $t_i$ " are reported. Assuming that the robber can move at most one unit per time step, your program must try to find the exact position of the robber at each time step.

## Input

The input contains the description of several robberies. The first line of each description consists of three numbers  $W, H, t$  ( $1 \leq W, H, t \leq 100$ ) where  $W$  is the width,  $H$  the height of the city and  $t$  is the time during which the city is locked.

The next contains a single integer  $n$  ( $0 \leq n \leq 100$ ), the number of messages the inspector received. The next  $n$  lines (one for each of the messages) consist of five integers  $t_i, L_i, T_i, R_i, B_i$  each. The integer  $t_i$  is the time at which the observation has been made ( $1 \leq t_i \leq t$ ), and  $L_i, T_i, R_i, B_i$  are the left, top, right and bottom respectively of the (rectangular) area which has been observed. ( $1 \leq L_i \leq R_i \leq W, 1 \leq T_i \leq B_i \leq H$ ; the point  $(1, 1)$  is the upper left hand corner, and  $(W, H)$  is the lower right hand corner of the city.) The messages mean that the robber was not in the given rectangle at time  $t_i$ .

The input is terminated by a test case starting with  $W=H=t=0$ . This case should not be processed.

## Output

For each robbery, first output the line "Robbery #k:", where  $k$  is the number of the robbery. Then, there are three possibilities:

If it is impossible that the robber is still in the city considering the messages, output the line "The robber has escaped."

In all other cases, assume that the robber really is in the city. Output one line of the form "Time step: The robber has been at  $x, y$ ." for each time step, in which the exact location can be deduced. ( $x$  and  $y$  are the column resp. row of the robber in time step.) Output these lines ordered by time.

If nothing can be deduced, output the line "Nothing known." and hope that the inspector will not get even more angry.

Output a blank line after each processed case.

## Sample Input

```
4 4 5
4
1 1 1 4 3
1 1 1 3 4
4 1 1 3 4
4 4 2 4 4
```

```
10 10 3
1
2 1 1 10 10
0 0 0
```

## Sample Output

```
Robbery #1:
Time step 1: The robber has been at 4,4.
Time step 2: The robber has been at 4,3.
Time step 3: The robber has been at 4,2.
Time step 4: The robber has been at 4,1.

Robbery #2:
The robber has escaped.
```

## Problem Source

Mid—Central European Regional Contest 1999

### 【题目大意】

检查员 Robstop 非常生气。昨晚，一家银行被盗，劫匪还未归案。这是今年发生的第三起抢劫案。尽管他竭尽全力阻止劫匪：尽快封锁所有通向城外的道路，使劫匪无法逃脱。然后检查员要求全城的人密切注意劫匪，但是得到的唯一回答是 “We don't see him.”。

但是这次，他已经有办法了！检查员 Robstop 决定分析劫匪是如何逃脱的。为了完成这个任务，他请你编写程序。根据检查员搜集到的信息，确定什么时间劫匪在什么地方。

非常巧合，发生银行抢劫案的城市是长方形。离开城市的道路被封闭一段时间  $t$ ，在这期间，某些观察点报告 “The robber isn't in the rectangle  $R_i$  at time  $t_i$ ”。假设，每单位时间劫匪最多能移动一个单位距离。在每个时间段，程序输出劫匪的精确位置。

### 输入格式

输入多个抢劫案件。第一行有三个数  $W, H, t$  ( $1 \leq W, H, t \leq 100$ )， $W$  是城市的宽度， $H$  是城市的高度， $t$  是封锁城市的时间。

然后是一个整数  $n$  ( $0 \leq n \leq 100$ )，检查员获得报告的数量。接下来  $n$  行（每行一条信息），每行五个整数  $t_i, L_i, T_i, R_i, B_i$ 。整数  $t_i$  ( $1 \leq t_i \leq t$ ) 是观察点报告的时间； $L_i, T_i, R_i, B_i$  是观察点（矩形）的左边、上边、右边和下边。（ $1 \leq L_i \leq R_i \leq W, 1 \leq T_i \leq B_i \leq H$ ；点  $(1, 1)$  是城市的左上角， $(W, H)$  是城市的右下角。）信息的意思是，在时间  $t_i$  劫匪不在该矩形里。

输入  $W=H=t=0$  结束，这组数据不需处理。

### 输出格式

对每个劫匪，先输出 “Robbery #k:”， $k$  是抢劫案件的编号。然后，有三种可能：

如果劫匪已经不在城市里，输出 “The robber has escaped.”。

对于其他所有的测试例，假设劫匪还在城市中。对每个时间段，输出一行：“Time step: The robber has been at  $x, y$ 。”。输出劫匪所处的确切位置。（ $x$  和  $y$  是劫匪的位置坐标。）按时间顺序输出。

如果不能推断出什么结果，输出 “Nothing known.”，希望检查员不会再烦恼。

每组测试数据后输出一空行。

## 【算法分析】

本题是根据给定的信息，模拟劫匪逃跑的各种可能性，最后分析劫匪的状态。

### （1）数据结构

在每一时刻，城市的状态都是不一样的，因此需要使用三维数组表示城市的状态：

```
char city[100][100][100];
```

第一维是时间，后面二维是城市平面图。只有两个状态，字符型数据就可以：

有劫匪时，单元的值 0；没有劫匪时，单元的值 1。

### （2）模拟劫匪逃跑的各种可能性

根据给定的信息进行推理。按时间顺序，需要进行两个方面的推理：

- 根据时间  $t_i-1$  的城市平面图，推理  $t_i$  时间的平面图（往前推理）；
- 根据时间  $t_i+1$  的城市平面图，推理  $t_i$  时间的平面图（往后推理）。

在推理时，掌握如下原则：

如果前一时间，单元  $(i, j)$  及其上下左右都没有劫匪，则当前时间单元  $(i, j)$  就没有劫匪。否则，只要单元  $(i, j)$  及其上下左右的任一处有劫匪，则单元  $(i, j)$  就有劫匪。

### （3）根据城市平面图，分析劫匪的状态

劫匪有三种状态，用变量 `escape` 表示：

escape 的值	表示的状态
0	劫匪已经不在城市里
1	不能推断出什么结果，即位置太多
2	劫匪还在城市中，输出劫匪的坐标

注意：这三种状态是互斥的。例如劫匪开始在城市里，后来又不能确定劫匪的状态，不能输出两个答案，只要输出劫匪在城市里的状态即可。

判定方法：

- 劫匪已经不在城市里：平面图上没有劫匪；
- 不能推断出什么结果：劫匪的位置超过两个，不能确定哪一个位置是正确的。

如果前面输出过劫匪的位置，则上面两种状态不会出现。

## 【程序代码】

程序名称： zju1144.c

题 目： Robbery

提交语言： C

运行时间： 20ms

运行内存： 1136KB

```
#include <stdio.h>
#include <memory.h>

char city[100][100][100]; //每一时刻的城市平面图
int W, H, t; //城市的宽度，高度和封锁城市时间
int n; //消息的条数
int main()
```

```

{
    int iCase = 0;                //抢劫案件的编号
    int escape;                  //劫匪的状态
    int robber;
    while(scanf("%d%d%d", &W, &H, &t) && (W||H||t))
    {
        memset(city, 0, sizeof(city));
        scanf("%d ", &n);
        int ti;                  //收到信息的时间
        int left, top, right, bottom; //矩形区域
        int i, j;
        int x, y, z;
        //读取n条信息
        for(i=0; i<n; i++)
        {
            scanf("%d%d%d%d", &ti, &left, &top, &right, &bottom);
            //没有劫匪的区域
            for (y = top-1; y <= bottom-1; y++)
                for(x = left-1; x <= right-1; x++)
                    city[ti-1][y][x] = 1;
        }
        //根据时间  $t_i-1$  的城市平面图, 推理  $t_i$  时间的平面图 (往前推理)
        for (z = 1; z < t; z++)
            for(j = 0; j < H; j++)
                for(i = 0; i < W; i++)
                {
                    robber = 1;
                    if (city[z-1][j][i]==0) robber=0;
                    if (j>0 && city[z-1][j-1][i]==0) robber=0;
                    if (i>0 && city[z-1][j][i-1]==0) robber=0;
                    if (j<H-1 && city[z-1][j+1][i]==0) robber=0;
                    if (i<W-1 && city[z-1][j][i+1]==0) robber=0;
                    //单元本身及其上下左右都没有劫匪时
                    if (robber==1) city[z][j][i] = 1;
                }
        //根据时间  $t_i+1$  的城市平面图, 推理  $t_i$  时间的平面图 (往后推理)
        for (z=t-2; z>=0; z--)
            for(j = 0; j<H; j++)
                for(i = 0; i<W; i++)
                {
                    robber = 1;
                    if (city[z+1][j][i]==0) robber=0;
                    if (j>0 && city[z+1][j-1][i]==0) robber=0;
                    if (i>0 && city[z+1][j][i-1]==0) robber=0;
                    if (j<H-1 && city[z+1][j+1][i]==0) robber=0;
                    if (i<W-1 && city[z+1][j][i+1]==0) robber=0;
                    //单元本身及其上下左右都没有劫匪时
                    if (robber==1) city[z][j][i] = 1;
                }
    }
}

```

```

    }
    printf("Robbery #d:\n", ++iCase);
    escape = 0;
    int mx, my;
    //对每一个单位时间
    for(i=0; i<t; i++)
    {
        robber = 0;
        //搜索城市平面图
        for (y=0; y<H; y++)
            for(x=0; x<W; x++)
                if (city[i][y][x]==0) //有劫匪
                    if (robber==0) //第一个有劫匪的单元
                    {
                        robber = 1;
                        if (escape!=2) escape=1;
                        mx = x, my = y; //劫匪的坐标
                    }
                else robber = 2; //劫匪的位置多于一个
        if (robber==1) //劫匪的位置只有一个
        {
            escape = 2; //劫匪的最终状态
            printf("Time step %d: The robber has been at %d,%d.\n",
                i+1, mx+1, my+1);
        }
    }
    if (escape==0) printf("The robber has escaped.\n");
    if (escape==1) printf("Nothing known.\n");
    printf("\n");
}
return 0;
}

```

## ZJU1146-LC-Display<sup>[1、2、3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

A friend of you has just bought a new computer. Until now, the most powerful computer he ever used has been a pocket calculator. Now, looking at his new computer, he is a bit disappointed, because he liked the LC-display of his calculator so much. So you decide to write a program that

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1146>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1102>

[3] <http://acm.uva.es/p/v7/706.html>

displays numbers in an LC-display-like style on his computer.

## Input

The input contains several lines, one for each number to be displayed. Each line contains two integers  $s, n$  ( $1 \leq s \leq 10$ ,  $0 \leq n \leq 99\,999\,999$ ), where  $n$  is the number to be displayed and  $s$  is the size in which it shall be displayed.

The input file will be terminated by a line containing two zeros. This line should not be processed.

## Output

Output the numbers given in the input file in an LC-display-style using  $s$  "-" signs for the horizontal segments and  $s$  "|" signs for the vertical ones. Each digit occupies exactly  $s+2$  columns and  $2s+3$  rows. (Be sure to fill all the white space occupied by the digits with blanks, also for the last digit.) There has to be exactly one column of blanks between two digits.

Output a blank line after each number. (You will find a sample of each digit in the sample output.)

## Sample Input

```
2 12345
3 67890
0 0
```

## Sample Output

```

      --  --  --
      |  |  |  |  |
      |  |  |  |  |
      --  --  --  --
      |  |  |  |  |
      |  |  |  |  |
      --  --  --
  ---  ---  ---  ---  ---
  |  |  |  |  |  |  |
  |  |  |  |  |  |  |
  |  |  |  |  |  |  |
  ---  ---  ---
  |  |  |  |  |  |  |
  |  |  |  |  |  |  |
  |  |  |  |  |  |  |
  ---  ---  ---  ---
```

## Problem Source

Mid—Central European Regional Contest 1999



## 【题目大意】

你的一位朋友刚买了一台新计算机。到目前为止，他用过的功能最强大的计算机是一台袖珍计算器。看着自己的新计算机，他有点失望，因为他喜欢计算器的 LC 显示器。你决定编写一个程序，在计算机上显示具有 LC 显示器风格的数字。

### 输入格式

输入有多行，每行是要显示的数。每行两个整数  $s, n$  ( $1 \leq s \leq 10, 0 \leq n \leq 99\,999\,999$ )， $n$  是要显示的数字， $s$  是显示数字时的尺寸。

当一行是两个零时，输入结束，这行不需处理。

### 输出格式

输出该数的 LC 显示器的风格。 $s$  个 “—” 用来表示水平线段尺寸， $s$  个 “|” 用来表示垂直线段尺寸。每个数字占用  $s+2$  列和  $2s+3$  行。（数字里其他的地方需用空格表示，最后一个数字也是这样。）两个数之间有一列空格。

每行数字显示之后有一空行。（每个数字如输出样例所示。）

## 【算法分析】

将数字以 LC 显示器的风格显示出来。题目可以归结为字符串处理或者模拟运算。

### （1）LC 显示器的数据结构

学习过汇编语言程序设计的读者都知道，这实际上是七段数码管的格式，如图 2-1 所示。数字 0~9 的七段数码管编码如图 2-2 所示。

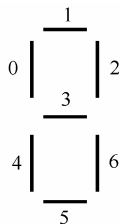


图 2-1 七段数码管的编号

七段数码管编号		0	1	2	3	4	5	6
数字	0	1	1	1	0	1	1	1
1	0	0	1	0	0	0	0	1
2	0	1	1	1	1	1	1	0
3	0	1	1	1	0	1	1	1
4	1	0	1	1	0	0	0	1
5	1	1	0	1	0	1	1	1
6	1	1	0	1	1	1	1	1
7	0	1	1	0	0	0	0	1
8	1	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1	1

图 2-2 七段数码管的编码

该表存储在数组中：

```
const int LCD[10][7];
```

### （2）构造显示矩阵

把显示结果构造到数组中，打印起来就十分方便：

```
char data[150][50];
```

对每一个数字，按七段数码管的格式填充数组，由函数实现：

```
void build(int x);
```

形参  $x$  就是要显示的数字。

数码管显示“|”时，用数字 1 表示，显示“—”时，用数字通信表示。由于每个数字占用  $s+2$  列，还有一个空列，使用变量 `len` 表示列（初值为 0），则有：

```
len += size + 3;
```

构造时，是按列行的顺序构造的。

（3）输出显示矩阵

输出时，是按行列的顺序。

## 【程序代码】

---

程序名称： zju1146.c

题    目： LC-Display

提交语言： C

运行时间： 0ms

运行内存： 168KB

---

```
#include <stdio.h>
#include <string.h>
//七段数码管的编码
const int LCD[10][7] = {
    {1, 1, 1, 0, 1, 1, 1},
    {0, 0, 1, 0, 0, 0, 1},
    {0, 1, 1, 1, 1, 1, 0},
    {0, 1, 1, 1, 0, 1, 1},
    {1, 0, 1, 1, 0, 0, 1},
    {1, 1, 0, 1, 0, 1, 1},
    {1, 1, 0, 1, 1, 1, 1},
    {0, 1, 1, 0, 0, 0, 1},
    {1, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 0, 1, 1}
};
int len; //每个数字的起始列号
int size; //水平线段/垂直线段的长度
char data[150][50]; //显示矩阵
char num[20]; //显示的数字

//构造显示矩阵
void build(int x){
    int i;
    if (LCD[x][0] == 1){ //0 号数码管
        for (i = 1; i <= size; ++i)
            data[len][i] = 1;
    }
    if (LCD[x][1] == 1){ //1 号数码管
        for (i = len + 1; i <= len + size; ++i)
            data[i][0] = 2;
    }
}
```

```

        if (LCD[x][2] == 1){                                //2 号数码管
            for (i = 1; i <= size; ++i)
                data[len + size + 1][i] = 1;
        }
        if (LCD[x][3] == 1){                                //3 号数码管
            for (i = len + 1; i <= len + size; ++i)
                data[i][size + 1] = 2;
        }
        if (LCD[x][4] == 1){                                //4 号数码管
            for (i = size + 2; i <= 2 * size + 1; ++i)
                data[len][i] = 1;
        }
        if (LCD[x][5] == 1){                                //5 号数码管
            for (i = len + 1; i <= len + size; ++i)
                data[i][2 * size + 2] = 2;
        }
        if (LCD[x][6] == 1){                                //6 号数码管
            for (i = size + 2; i <= 2 * size + 1; ++i)
                data[len + size + 1][i] = 1;
        }
        //计算下一个显示数字的位置
        len += size + 3;
    }
    int main(){
        int i, j;
        while(scanf("%d%s", &size, num) && size){
            len = 0;
            memset(data, 0, sizeof(data));
            //构造显示矩阵
            for (i = 0; i<strlen(num); ++i)
                build(num[i] - '0');
            //输出显示矩阵, 注意行列编号
            for (i = 0; i<=2*size+2; ++i){
                for (j = 0; j<len-1; ++j){
                    if (data[j][i] == 1) printf("|");
                    else if (data[j][i] == 2) printf("-");
                    else printf(" ");
                }
                printf("\n");
            }
            printf("\n");
        }
        return 0;
    }

```

# ZJU1153-Tournament Seeding<sup>[1, 2, 3]</sup>

Time Limit: 10 Seconds

Memory Limit: 32768KB

In many competitions, players are given seeding numbers to represent their relative strength, where the best player is given seed #1, the second best is given seed #2, etc.. The seeding of a single elimination tournament is an arrangement of matches which keeps the best players from playing each other until as late as possible (the last round with players seeded 1 and 2). Define the strength for a match as the sum of the two seeds in the match, and the ideal strength as the strength for the match assuming the best two players make it to that match (or in other words, the minimum sum of the two seeds which could play in that match). The total number of rounds  $r$  needed for a tournament of  $n$  contestants is given by the formula  $r = \lceil \log_2(n) \rceil$ . If we call the finals of the tournament round  $r$ , the semi-finals round  $r-1$ , etc., then the way in which seeding is done can be described as follows: in round  $k$ , we arrange players such that the ideal match strength for all matches in that round is  $2^{(r-k+1)} + 1$ . For example, the seeding of a match of 13 people would look as follows:

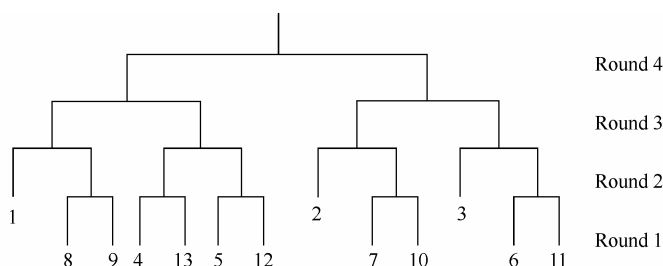


Figure 2-3

You are asked to solve the following problem: given the values of  $n$  and  $m$ , determine the earliest round in a tournament of  $n$  players in which a match strength of  $m$  could occur. You may assume that the seeding is done in the manner described above.

## This problem contains multiple test cases!

The first line of a multiple input is an integer  $N$ , then a blank line followed by  $N$  input blocks. Each input block is in the format indicated in the problem description. There is a blank line between input blocks.

The output format consists of  $N$  output blocks. There is a blank line between output blocks.

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1153>

[2] <http://acm.tju.edu.cn/toj/showp1155.html>

[3] <http://plg1.cs.uwaterloo.ca/~acm00/regionals/>

## Input

You will be given a number of cases in the input. Each case will be specified on one line, consisting of two integers  $n$  and  $m$ , where  $2 \leq n \leq 100$  and  $m \geq 3$ . You may assume in each case that a match strength of  $m$  can occur during some round. Input is terminated by a case in which  $n=m=0$ .

## Output

For each test case, print the case number and the earliest round for that case in format shown below.

## Sample Input

```
1
100 3
13 10
0 0
```

## Sample Output

```
Case 1: Round 7
Case 2: Round 2
```

## Problem Source

East Central North America 1999, Practice

### 【题目大意】

在一些竞赛中，竞赛者会有种子号码来表示他们的能力，最有实力的竞赛者得到 1 号，第二强的竞赛者得到 2 号，等等。在单淘汰赛中，采用种子号码安排比赛，能让最强的竞赛者在最后面比赛（最后一轮比赛由 1 号对 2 号）。一场比赛的强度定义为两个竞赛者的种子号码之和。理想的强度可以假设两个最强的竞赛者出现在一场比赛中（或者换句话说，两个竞赛者的号码之和是最小的出现在同一场比赛）。有  $n$  个竞赛者，总的循环圈数用  $r$  表示，则  $r = \text{ceiling}(\log(2, n))$ （不小于给定实数的最小整数）。如果我们说决赛在第  $r$  圈，则半决赛在第  $r-1$  圈，等等。安排种子号码的方式可以描述为：在  $k$  圈中，以便每场比赛都是理想的强度，我们安排竞赛选手的强度为： $2^{(r-k+1)} + 1$ 。例如：比赛中有 13 个竞赛者，种子号码的安排如图 2-3 所示。

### 编程任务

给定  $n$  和  $m$  的值，在有  $n$  个竞赛者的比赛中，确定产生比赛强度  $m$  的最早的圈。你可以假设种子号码是按上述方式安排的。

本题包含多组测试例！

多组测试例的第一行是一个整数  $N$ ，然后是一个空行，接着是  $N$  个输入数据块。每个数据块的格式在问题描述中给出。每个数据块之间有一个空行。

输出格式包括  $N$  个输出数据块，每个输出数据块之间有一个空行。

### 输入格式

输入数据中有多组测试例，每个测试例一行，是两个整数  $n$  和  $m$ ， $2 \leq n \leq 100$  和  $m \geq 3$ 。在每个测试例中，你可以假设比赛强度  $m$  会产生在某一圈中。当  $n$  和  $m$  都为 0 时，表示输入

结束。

### 输出格式

对每个测试例，输出测试例的编号和最早的圈，输出格式如样例所示。

### 【算法分析】

观察种子号码的安排图，看似像一棵二叉树。二叉树的叶子结点是种子号码，二叉树的深度是比赛的圈数。一场比赛的强度定义为两个竞赛者的种子号码之和。题目要求确定产生比赛强度  $m$  的最早的圈。

#### (1) 数据结构

存储按位置编号的种子号码数值：

```
int match[128];
```

对于 16 个竞赛者，二叉树叶子结点的值如图 2-4 所示：

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
match[i]	1	16	8	9	4	13	5	12	2	15	7	10	3	14	6	11

图 2-4 16 个竞赛者的二叉树叶子结点的值

存储种子号码对应位置的数值：

```
int play[129];
```

实际上是数组 `match` 的逆序值，如图 2-5 所示：

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
play[i]	0	8	12	4	6	14	10	2	3	11	15	7	5	13	9	1

图 2-5 数组 `play` 是数组 `match` 的逆序值

由于竞赛人数  $n$  ( $2 \leq n \leq 100$ ) 不多，在题目中一次性产生了数组 `match` 和数组 `play` 的值。产生数组 `match` 的算法：

总的循环圈数用  $r$  ( $1 \leq r \leq 7$ ) 表示，则每场比赛理想的强度是：

$$2^{r-k+1} + 1, (k=1 \sim r)$$

即位置  $j$  ( $0 \leq j \leq 127$ )，跨度为  $w$  的一对选手，种子号码为：

$$\text{match}[j] + \text{match}[j + w] = 2^{r-k+1} + 1$$

上式中， $r-k+1=1 \sim r$ 。在程序中由外循环  $i$  ( $i=1 \sim 7$ ) 实现。

#### (2) 根据竞赛者数量 $n$ ，确定比赛的圈数 $r$

可以根据题目中的公式计算：

$$r = \text{ceiling}(\log(2, n))$$

在程序中，是通过对  $n$  不断除以 2 并计数实现的。

#### (3) 搜索产生比赛强度 $m$ 的最早的圈 $\min$

由于数组 `match` 是一次性按 128 人产生的，所以当  $n < 128$  时，实际的种子号码要在数组 `match` 中查找。

选手 1 的强度为  $I$  ( $1 \leq i \leq (m-1)/2$ )，位置为  $p_1$ ；选手 2 的强度为  $m-i$ ，位置为  $p_2$ 。跨度  $y$  随着圈数的增加而成倍增加，然后根据跨度值修正位置  $p_1$  和  $p_2$ 。

对不同的强度  $i$ ，在有效范围内 ( $i \leq n$  且  $(m-I) \leq n$ ) 从二叉树的叶子结点向根部搜索，直到  $p_1 = p_2$  (实现了比赛强度为  $m$  的比赛)，得到比赛的圈数。按题目要求输出最小的圈数。

## 【程序代码】

---

程序名称: zju1153.c  
题 目: Tournament Seeding  
提交语言: C  
运行时间: 30ms  
运行内存: 160KB

---

```
#include <stdio.h>
#include <memory.h>

int match[128];           //按位置编号的种子号码数值
int play[129];           //按种子号码的位置数值

int main() {
    int i, j, k;
    int block;            //数据块的数量
    int n, m;             //竞赛人数, 比赛强度
    //计算按位置编号的种子数值
    int w = 64;
    int r = 2;
    match[0] = 1;
    for( i = 1; i <= 7; i++) {
        for( j = 0; j < 128; j += 2*w )
            match[j+w] = r+1-match[j];
        w /= 2;
        r *= 2;
    }
    //计算按种子编号的位置数值
    for( i = 0; i < 128; i++ )
        play[match[i]] = i;

    scanf("%d", &block);
    for (k=0; k<block; k++) {
        if (k) printf("\n");

        int iCase = 1;           //测试例数
        int p1, p2;             //进入比赛的两个种子
        int round;              //比赛的圈数
        int min;                //找到 m 的最早的圈数
        int x, y;
        while (scanf("%d%d", &n, &m ) && (n || m)) {
            //根据竞赛者数量, 确定比赛的圈数
            int r = 0;           //比赛的圈数
            y = n-1;
            while( y > 0 ) {
                y /= 2;
```

```

        r++;
    }
    x = 1<<(7-r);
    min = 8; //因为最多是 7 圈，所以 8 就是∞
    //搜索产生比赛强度 m 的最早的圈
    for( i = 1; i <= (m-1)/2; i++ )
        if( i <= n && (m-i) <= n ) //在有效范围内搜索
        {
            round = 0; //当前跨度内的圈数
            y = 2*x; //当前跨度
            p1 = play[i]; //选手 1（强度为 i）的位置
            p2 = play[m-i]; //选手 2（强度为 m-i）的位置
            do {
                //根据跨度修正位置
                p1 = p1-(p1%y);
                p2 = p2-(p2%y);
                //圈数上升 1 层，跨度增加 1 倍
                y *= 2;
                round++;
            } while (p1 != p2);
            if( round < min ) min = round;
        }
    printf( "Case %d: Round %d\n", iCase++, min );
}
}
return 0;
}

```

## ZJU1160-Biorhythms<sup>[1、2、3]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768KB

---

Some people believe that there are three cycles in a person's life that starts the day he or she is born. These three cycles are the physical, emotional, and intellectual cycles, and they have periods of lengths 23, 28, and 33 days, respectively. There is one peak in each period of a cycle. At the peak of a cycle, a person performs at his or her best in the corresponding field (physical, emotional or mental). For example, if it is the mental curve, thought processes will be sharper and concentration will be easier.

Since the three cycles have different periods, the peaks of the three cycles generally occur at

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1160>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1006>

[3] <http://acm.uva.es/p/v7/756.html>



different times. We would like to determine when a triple peak occurs (the peaks of all three cycles occur in the same day) for any person. For each cycle, you will be given the number of days from the beginning of the current year at which one of its peaks (not necessarily the first) occurs. You will also be given a date expressed as the number of days from the beginning of the current year. Your task is to determine the number of days from the given date to the next triple peak. The given date is not counted. For example, if the given date is 10 and the next triple peak occurs on day 12, the answer is 2, not 3. If a triple peak occurs on the given date, you should give the number of days to the next occurrence of a triple peak.

## This problem contains multiple test cases!

The first line of a multiple input is an integer  $N$ , then a blank line followed by  $N$  input blocks. Each input block is in the format indicated in the problem description. There is a blank line between input blocks.

The output format consists of  $N$  output blocks. There is a blank line between output blocks.

## Input

You will be given a number of cases. The input for each case consists of one line of four integers  $p$ ,  $e$ ,  $i$ , and  $d$ . The values  $p$ ,  $e$ , and  $i$  are the number of days from the beginning of the current year at which the physical, emotional, and intellectual cycles peak, respectively. The value  $d$  is the given date and may be smaller than any of  $p$ ,  $e$ , or  $i$ . All values are non-negative and at most 365, and you may assume that a triple peak will occur within 21252 days of the given date. The end of input is indicated by a line in which  $p=e=i=d=-1$ .

## Output

For each test case, print the case number followed by a message indicating the number of days to the next triple peak, in the form:

Case 1: the next triple peak occurs in 1234 days.

Use the plural form "days" even if the answer is 1.

## Sample Input

```
1
0 0 0 0
0 0 0 100
5 20 34 325
4 5 6 7
283 102 23 320
203 301 203 40
-1 -1 -1 -1
```

## Sample Output

Case 1: the next triple peak occurs in 21252 days.

Case 2: the next triple peak occurs in 21152 days.  
Case 3: the next triple peak occurs in 19575 days.  
Case 4: the next triple peak occurs in 16994 days.  
Case 5: the next triple peak occurs in 8910 days.  
Case 6: the next triple peak occurs in 10789 days.

## Problem Source

East Central North America 1999; Pacific Northwest 1999

### 【题目大意】

有些人相信，人从出生开始就有三个生物周期。这三个生物周期分别是体力，情绪和智力，周期分别为 23, 38 和 33 天。在每个周期里都有一个高潮。在一个周期中高潮的时候，人们在相应的方面（体力，情绪或智力）达到最佳状态。例如，如果是心理曲线达到高潮，思考将变得睿智，注意力更容易集中。

由于这三个生物周期都不同，所以它们产生高潮的时间都是不同的。我们想要确定一个人什么时候这三个高潮将会同时产生（这三个高潮产生在同一天）。已知每个生物周期上次高潮产生时距离年初的天数（不一定是第一次产生），以及从年初开始的一个天数（起始时间）。你的任务是求出下次三个生物周期高潮同时产生时，距离起始时间的天数，不包括起始时间那天。例如，起始时间的天数是 10，而下次三个生物周期高潮同时产生的时间为 12，则答案是 2 而不是 3。如果三个生物周期高潮同时产生时就在起始时间那天，你要计算出下一次三个生物周期高潮同时产生的天数。

### 本题包含多组测试例！

多组测试例的第一行是一个整数  $N$ ，然后是一个空行，接着是  $N$  个输入数据块。每个数据块的格式在问题描述中给出。每个数据块之间有一个空行。

输出格式包括  $N$  个输出数据块，每个输出数据块之间有一个空行。

### 输入格式

输入有多组测试例。每个测试例一行，是 4 个整数  $p$ ,  $e$ ,  $i$  和  $d$ 。数值  $p$ ,  $e$  和  $i$  分别表示体力，情绪和智力达到高峰时距离年初的天数。数值  $d$  是给定的日期，也许小于  $p$ ,  $e$  或  $i$ 。所有数值都是非负的，小于等于 365。你可以假定三个生物周期高潮同时产生时，距离起始时间的天数在 21252 天之内。当一行  $p=e=i=d=-1$  时，表示输入结束。

### 输出格式

对每一个测试例，输出测试例编号，接着是下一次三个生物周期高潮同时产生的时间，格式如下

Case 1: the next triple peak occurs in 1234 days.

即使答案只有一天，也用复数形式 “days” 表示。

### 【算法分析】

以第三个样例数据为例：

5 20 34 325

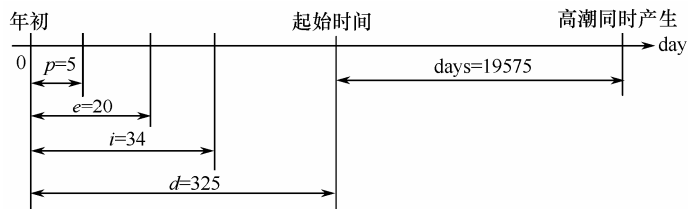


图 2-6 各个数据之间的相对关系

图 2-6 中，变量  $p$ ， $e$ ， $i$  和  $d$  的顺序是任意的，而高潮同时产生的时间（days）肯定是从起始时间（d）开始计算。

简单的办法就是令  $\text{days}=1$ ，从起始时间的第二天开始计算（避免起始时间这天），然后每一天往上搜，直到发现满足要求的那一天。

关系表达式：

$(\text{days}+\text{d}-\text{p}) \% 23 \parallel (\text{days}+\text{d}-\text{e}) \% 28 \parallel (\text{days}+\text{d}-\text{i}) \% 33$

表示在三项同时为 0 时结果为 0，即高潮同时产生的时间（days）找到了。

### 【程序代码】

程序名称： zju1160.c

题 目： Biorhythms

提交语言： C

运行时间： 210ms

运行内存： 160KB

```
#include <stdio.h>

int main()
{
    int p, e, i;           //体力，情绪和智力达到高峰时距离年初的天数
    int d;                 //给定的日期，是从年初开始的一个天数

    int N;                 //数据块的数量
    scanf("%d", &N);
    while (N--)
    {
        int cases = 0;     //测试例数
        while (scanf("%d%d%d%d", &p, &e, &i, &d) && p != -1) {
            {
                days = 1;   //避免刚好在起始时间（days=0）那天
                //从每一天往上搜，直到发现满足要求的那一天
                while ((days+d-p) % 23 || (days+d-e) % 28 || (days+d-i) % 33) days++;
                printf( "Case %d: ", ++cases);
                printf( "the next triple peak occurs in %d days.\n", days);
            }
            if (N) printf("\n");
        }
    }
}
```

```
    return 0;
}
```

### 【算法改进】

由于每一天往上计算比较费时，利用同余的特性能够很快计算出答案：

- 计算  $p$  和  $e$  同余的那一天：

```
while ((p-e)%28 != 0)    p += 23;
```

- 计算  $p$  和  $i$  同余的那一天（注意，增长的天数是  $23 \times 28$ ，即共同的周期）：

```
while ((p-i)%33 != 0)    p += 23*28;
```

- 计算  $p$  和  $d$  关系（注意：增加或减少的天数是  $23 \times 28 \times 33$ ，以确保  $days$  刚好在  $d$  的右边）：

```
while (p > d)    p -= 23*28*33;
```

```
while (p <= d)    p += 23*28*33;
```

参考代码：zju1160-01.c

在East Central North America 1999 的比赛网站<sup>[1]</sup>上有标程。

---

[1] <http://plg1.cs.uwaterloo.ca/~acm00/regionals/>

## 第三章 字符串处理题

### ZJU1109-Language of FatMouse<sup>[1]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768KB

---

We all know that FatMouse doesn't speak English. But now he has to be prepared since our nation will join WTO soon. Thanks to Turing we have computers to help him.

#### Input Specification

Input consists of up to 100005 dictionary entries, followed by a blank line, followed by a message of up to 100005 words. Each dictionary entry is a line containing an English word, followed by a space and a FatMouse word. No FatMouse word appears more than once in the dictionary. The message is a sequence of words in the language of FatMouse, one word on each line. Each word in the input is a sequence of at most 10 lowercase letters.

#### Output Specification

Output is the message translated to English, one word per line. FatMouse words not in the dictionary should be translated as "eh".

#### Sample Input

```
dog ogday
cat atcay
pig igpay
froot ootfray
loops oopslay
```

```
atcay
ittenkay
oopslay
```

#### Output for Sample Input

```
cat
eh
loops
```

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1109>

## Problem Source:

Zhejiang University Training Contest 2001

### 【题目大意】

我们都知道 FatMouse 不会说英语。因为我国很快要加入世界贸易组织，所以它现在不得不准备起来。非常感谢 Turing，现在我们可以用计算机来帮助它。

### 输入格式

输入多达 100 005 个词典条目，接着是一个空行，然后是多达 100005 个信息。每个词典条目占一行，包含一个英语单词，一个空隔和一个 FatMouse 单词。词典中每个 FatMouse 词出现一次。信息是一组 FatMouse 语言的单词，每行一个单词。输入的每个单词最多 10 个小写字母。

### 输出格式

输出用英语单词表达的信息，每行一个单词。如果 FatMouse 单词未出现在词条中，就输出“eh”。

### 【算法分析】

本题主要是字符串匹配。将每条 FatMouse 单词转换成用英语表示的单词。

如果使用 Trie 树算法，速度会很快（130ms），但编程复杂，即使竞赛时想到了，如果没有准备模板，编程也是比较费时的。

如果采用枚举的方法，算法时间复杂度是  $O(n^2)$ ，容易超时。

为了加快速度，这里采用先排序，然后二分查找。时间主要花在排序上面，采用快排函数 `qsort()`，其时间复杂度是  $O(n \lg n)$ 。

#### （1）数据结构

用结构体表示词典条目：

```
struct node{
    char english[11], mouse[11];
} data[100010];
```

FatMouse 单词：

```
char word[11];
```

#### （2）排序算法

采用 C 语言库函数 `qsort`：

```
qsort(data, n, sizeof(node), cmp);
```

其中变量  $n$  是词典条目的总数，`cmp` 是比较因子函数。

#### （3）二分搜索算法的实现

关于二分搜索算法的实现，请参看有关数据结构的书籍。

### 【程序代码】

---

程序名称：	zju1109-qsort.cpp
题    目：	Language of FatMouse
提交语言：	C++
运行时间：	300ms
运行内存：	4472KB

---

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

const int MAXN = 100010;
//表示词典条目
struct node{
    char english[11], mouse[11];
} data[MAXN];

//比较因子函数
int cmp(const void *a, const void *b){
    node *ta = (node *)a;
    node *tb = (node *)b;
    //按字符串序升序排序（不是字典序。字符串序中，相同长度的字符串在一起）
    return strcmp((*ta).mouse, (*tb).mouse);
}

int main(){
    char word[11]; // FatMouse 单词
    //构造词典
    char s[24];
    int n = 0; //词典条目的总数
    while (gets(s)){ //读取一行
        if (!strlen(s)) break; //是空行
        //从缓冲中读取词典条目
        sscanf(s, "%s%s", data[n].english, data[n].mouse);
        ++n;
    }
    qsort(data, n, sizeof(node), cmp); //排序
    n--;
    //二分搜索算法的实现
    int left, right, mid; //分别为左边界、右边界和中点
    int ok;
    //对每一条信息，即 FatMouse 语言的单词进行转换
    while (gets(word)){
        left = ok = 0;
        right = n;
        while (left <= right){ //没有查找完
            mid = (left + right) / 2;
            if (strcmp(data[mid].mouse, word)==0){ //找到了
                ok = 1;
                break;
            }
            //没有找到，修正右边界或左边界
            if (strcmp(data[mid].mouse, word)>0) right = mid - 1;
            if (strcmp(data[mid].mouse, word)<0) left = mid + 1;
        }
    }
}

```

```

    }
    //输出结果
    if (ok) printf("%s\n", data[mid].english);
    else printf("eh\n");
}
return 0;
}

```

## 【其他算法】

本题如果使用 C++ 标准模板库中的 `map()` 容器，就变得非常简单。  
将 `FatMouse` 单词作为 `key`，英语单词作为 `value`，如表 3-1 所示：

表 3-1 词典条目在 `map()` 容器中的表示

key	value
atcay	cat
igpay	pig
ogday	dog
oopslay	loops
ootfray	froot

每次读取一条信息作为 `map()` 容器的 `key`，输出相应的 `value` 就行了。由于标准模板库功能多，占用的资源多，运行时间就要慢一些。随着竞赛题目难度的不断提高，能够运用标准模板库解题的机会越来越少。

## 【程序代码】

---

程序名称： zju1109-map.cpp  
 题 目： Language of FatMouse  
 提交语言： C++  
 运行时间： 1300ms  
 运行内存： 9556KB

---

```

#include<iostream>
#include<map>
using namespace std;

int main(){
    map<string,string> entry;           //表示词典条目
    char line[30];
    char english[12],mouse[12];         //每个条目
    string value, key;
    map<string,string>::iterator location, pos; //map()容器的指针
    //构造词典
    while(gets(line)){
        if(strlen(line) == 0)break;      //空行
        sscanf(line, "%s%s", english, mouse); //读取词条
    }
}

```



```

        key = mouse;
        value = english;
        entry[key]=value;
    }
    //对每一条信息
    while(cin>>line){
        //在词典中查一查
        location = entry.find(line);
        if (location!=entry.end()) cout<<entry[line]<<endl;    //词典中有
        else cout<<"eh"<<endl;    //词典中没
    }
    return 0;
}

```

## ZJU1111-Poker Hands<sup>[1、2]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

A poker deck contains 52 cards — each card has a suit which is one of clubs, diamonds, hearts, or spades (denoted C, D, H, S in the input data). Each card also has a value which is one of 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, king, ace (denoted 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, A). For scoring purposes, the suits are unordered while the values are ordered as given above, with 2 being the lowest and ace the highest value.

A poker hand consists of 5 cards dealt from the deck. Poker hands are ranked by the following partial order from lowest to highest

- **High Card.** Hands which do not fit any higher category are ranked by the value of their highest card. If the highest cards have the same value, the hands are ranked by the next highest, and so on.

- **Pair.** 2 of the 5 cards in the hand have the same value. Hands which both contain a pair are ranked by the value of the cards forming the pair. If these values are the same, the hands are ranked by the values of the cards not forming the pair, in decreasing order.

- **Two Pairs.** The hand contains 2 different pairs. Hands which both contain 2 pairs are ranked by the value of their highest pair. Hands with the same highest pair are ranked by the value of their other pair. If these values are the same the hands are ranked by the value of the remaining card.

- **Three of a Kind.** Three of the cards in the hand have the same value. Hands which both contain three of a kind are ranked by the value of the 3 cards.

- **Straight.** Hand contains 5 cards with consecutive values. Hands which both contain a

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1111>

[2] <http://acm.uva.es/p/v103/10315.html>

straight are ranked by their highest card.

- **Flush.** Hand contains 5 cards of the same suit. Hands which are both flushes are ranked using the rules for High Card.

- **Full House.** 3 cards of the same value, with the remaining 2 cards forming a pair. Ranked by the value of the 3 cards.

- **Four of a kind.** 4 cards with the same value. Ranked by the value of the 4 cards.

- **Straight flush.** 5 cards of the same suit with consecutive values. Ranked by the highest card in the hand.

Your job is to compare several pairs of poker hands and to indicate which, if either, has a higher rank.

## Input Specification

Several lines, each containing the designation of 10 cards: the first 5 cards are the hand for the player named "**Black**" and the next 5 cards are the hand for the player named "**White**."

## Output Specification

For each line of input, print a line containing one of:

Black wins.

White wins.

Tie.

## Sample Input

```
2H 3D 5S 9C KD 2C 3H 4S 8C AH
2H 4S 4C 2D 4H 2S 8S AS QS 3S
2H 3D 5S 9C KD 2C 3H 4S 8C KH
2H 3D 5S 9C KD 2D 3H 5C 9S KH
```

## Sample Output

```
White wins.
Black wins.
Black wins.
Tie.
```

## Problem Source

University of Waterloo Local Contest 1998.06.06

### 【题目大意】

一副扑克牌 52 张，每张牌有花色：梅花（club），方块（diamond），红桃（heart）和黑桃（spade）（输入数据中用 C、D、H、S 表示）。每张牌也有一个数值，分别为 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, ace（输入中用 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, A 表示）。为了便于计分，不考虑牌的花色顺序，只考虑牌的数值顺序（2 最小，A 最大）。

梭哈游戏是每次发 5 张牌。梭哈游戏的计分方法，是根据扑克牌的部分顺序按下列规则从

最低分到最高分：

● **High Card: 散牌。**不满足任何更高条件的牌，就以他们各自的最大牌排名。如果有相同的最大牌，就比较次大的牌，依次类推。

● **Pair: 对子。**是指 5 张牌里有 2 张的值是相同的。如果双方都是对子，比对子的大小；如果对子也一样，比第二个对子或单张大小，以递减的顺序根据剩下牌的值得排名。

● **Two Pairs: 两对。**是指有两个对子。如果大家都是两对，比大对子的大小，如果大对子也一样，比小对子的大小。如果两对都相同，那么就按最后那张排名。

● **Three of a Kind: 三条。**是指有三张牌的值得相同。如果有这样的三张牌，那么就以这三张牌的值得排名。

● **Straight: 顺子。**是指 5 张牌的值得连续的。如果大家都是顺子，那么就比最大的一张牌。

● **Flush: 同花。**是指 5 张牌是同一种花色。如果大家都是同花，那么就以 High Card 的规则排名。

● **Full House: 葫芦。**是指 3 张牌的值得相同的，另 2 张成一对。那么就根据 3 张牌的值得排名。

● **Four of a kind: 铁支。**是指 4 张牌的值得相同的。那么就根据这 4 张牌的值得排名。

● **Straight flush: 同花顺。**是指同一种花色的顺子。那么就比最大的一张牌。

编程任务：给梭哈游戏当裁判，判定每副牌的最高得分。

#### 输入说明

输入多行。每行是 10 张牌：前 5 张是发给选手“Black”的，后 5 张是发给选手“White”的。

#### 输出说明

对输入的每一行，输出下列的一行：

Black wins。

White wins。

Tie。

#### 【算法分析】

这是一种类似于梭哈的游戏，梭哈游戏要考虑花色的大小，参看联众在线游戏<sup>[1]</sup>。

很容易想到的办法是模拟，根据规则分析最高分，然后比较两个选手得分的大小。在互联网上有“Accepted”的程序，如“yllfever的专栏<sup>[2]</sup>”和“hoodlum1980（發發）的技术博客<sup>[3]</sup>”，而且有代码分析，但是代码都很长。这里给出的算法是将字符运算转换为数字运算，虽然运用了解题技巧，但算法的复杂性就高了一些，程序的阅读理解也困难了一些。

##### （1）样例分析

前 5 张是“Black”的，后 5 张是“White”的。

①2H 3D 5S 9C KD 2C 3H 4S 8C AH

---

[1] <http://www.ourgame.com/game/game-intro/suoha.html>

[2] <http://blog.csdn.net/yllfever/archive/2008/07/01/2602911.aspx>

[3] <http://www.cnblogs.com/hoodlum1980/archive/2008/11/15/1319107.html>

全是散牌，就比大小。最大的牌：Black 是 K，而 White 是 A，White 赢。

②2H 4S 4C 2D 4H 2S 8S AS QS 3S

Black 有一个三条（4S 4C 4H）和一个对子（2H 2D），是葫芦，而 White 是同花（全是 S），Black 赢。

③2H 3D 5S 9C KD 2C 3H 4S 8C KH

全是散牌，就比大小。最大的 K 相等，次大的：Black 是 9，而 White 是 8，Black 赢。

④2H 3D 5S 9C KD 2D 3H 5C 9S KH

两家牌的值完全一样，平局。

## （2）数据结构

一副牌的值和花色：

```
char *deck="23456789TJQKA", *suit="CDHS";
```

扑克牌本来是字符串，这里转换成数字，存储方法如图 3-1 所示。

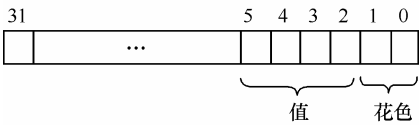


图 3-1 扑克牌的数字表示

扑克牌的值在存储时，要左移两位，读出时要右移两位。两家的牌用数组表示：

```
int deal[2][6];
```

从扑克牌值的角度看，相同的值计数为

```
int count[13];
```

梭哈游戏的规则，有 9 种情况（见表 3-2），表示为

```
int rank;
```

一手牌中不同值的个数为

```
int number[9]={5, 4, 3, 1, 1, 5, 2, 1, 1};
```

表 3-2 梭哈游戏的规则

规 则	变量 rank 的值	数组 number 的值	注 释
High Card: 散牌	0	5	5 张牌都不一样
Pair: 对子	1	4	有 4 张牌不一样
Two Pairs: 两对	2	3	有 3 张牌不一样
Three of a Kind: 三条	3	1	记三条中最大不相同的牌
Straight: 顺子	4	1	记顺子中最大不相同的牌
Flush: 同花	5	5	5 张牌都不一样
Full House: 葫芦	6	2	有 2 张牌不一样
Four of a kind: 铁支	7	1	记 4 张牌中最大不相同的牌
Straight flush: 同花顺	8	1	记顺子中最大不相同的牌

出现三条和铁支时，其他的牌不用考虑，所以也当作一个值。

（3）对两手牌分别计算 rank 和 number 的值

很多规则中要比牌的大小，因此在计算 rank 的时候，还要保存牌的大小：

```
int value[2][7];
```

对每一家的牌分别判断:

①由于从最大的牌比起, 所以要对扑克牌降序排序

```
qsort(deal[i], 5, sizeof(int), &compare), i=0, 1;
```

② 统计 5 张牌的数值重复的个数

统计结果放在数组 **count** 中。

③对扑克牌的 13 张进行枚举, 级别放在变量 **rank** 中

- 如果重复的数字是 2, 遇到 1 个对子, 2 个对子或者葫芦;
- 如果重复的数字是 3, 遇到 1 个条子或者葫芦;
- 如果重复的数字是 4, 就是铁支。

对 5 张不一样的牌, 还要判断是否是顺子、同花或者同花顺。

- 如果 5 张牌的花色都一样, 就是同花;

- 如果 5 张牌, 相邻牌的值差为 1 (因为已经排序), 则是顺子;

- 上面两个条件都满足, 就是同花顺。

④在进行级别判断时, 还要保存不同牌的值

⑤两家牌根据级别 **rank** 和牌的大小 **number** 进行比较, 决定胜负。

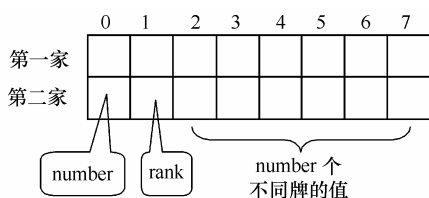


图 3-2 判断结束时, 数组 **value** 的状态

判断结束时, 存放在数组 **value** 中的值如图 3-2 所示。

## 【程序代码】

程序名称: zju1111.c

题目: Poker Hands

提交语言: C

运行时间: 0ms

运行内存: 160KB

```
#include <stdio.h>
#include <memory.h>
#include <stdlib.h>
```

```
char *deck="23456789TJQKA", *suit="CDHS"; //一副牌的值和花色
int number[9]={5, 4, 3, 1, 1, 5, 2, 1, 1}; //一手牌中不同值的个数
```

//排序因子, 降序

```
int compare(const void *a, const void *b)
{
    return(((int *)b))-(((int *)a)));
}
```

```
int main()
{
```

```

int deal[2][6]; //发给两家的牌
int value[2][7]; //两家牌的大小
int count[13]; //扑克牌中相同的值计数
int *p_deal; //指向数组 deal 一行的指针
int *p_value; //指向数组 value 一行的指针
int i, j;
char card[10]; //一张牌

while(1)
{
    //构造数组 deal, 将扑克牌转换为数字存储
    for(i=0; i<2; i++)
        for(j=0; j<5; j++)
        {
            if (scanf("%s", card)==EOF) return 0; //处理完毕
            deal[i][j] = ((strchr(deck, card[0])-deck)<<2)
                + (strchr(suit, card[1])-suit);
        }
    int rank; //梭哈游戏的规则
    int k; //某规则出现的次数
    memset(value, 0, sizeof(value));
    //分别处理每一家的牌
    for(i=0; i<2; i++)
    {
        qsort(deal[i], 5, sizeof(int), &compare); //降序排序
        p_deal = deal[i];
        p_value = value[i];
        memset(count, 0, sizeof(count));
        //统计 5 张牌的数值重复的个数
        for(j=0; j<5; j++)
            count[p_deal[j]>>2]++;
        rank=0;
        //分别处理每一张牌
        for(j=12; j>=0; j--)
            if(count[j]>1) //有重复的牌
            {
                switch(count[j])
                {
                    case 2: //有一个对子
                        switch(rank)
                        {
                            case 0: rank = 1; p_value[2] = j; break;
                                //第一个对子
                            case 1: rank = 2; p_value[3] = j; break;
                                //第二个对子
                            case 3: rank = 6; break;
                                //已经有一个三条

```

```

    }
    break;
case 3:                                     //有一个三条
    if(rank==0) rank = 3;                   //只有一个三条
    else rank = 6;                           //已经有一个对子
    p_value[2] = j;                           //记录该三条的值
    break;
case 4:                                     //有一个铁支
    rank = 7;
    p_value[2] = j;                           //记录该铁支的值
    break;
}
}
//因为不可能有 5 张牌一样的, 所以 count[j]=1
//对 5 张不一样的牌判断是否有顺子、同花或者同花顺
if(rank<6)
{
    k = 3;
    //判断花色: 如果有不一样的, 令 k 的第 0 位为 0
    for(j=1; j<5; j++)
        if((p_deal[j]&3) != (p_deal[0]&3)) {k &= 2; break;}
    //判断牌的大小: 如果有不一样的, 令 k 的第 1 位为 0
    for(j=1; j<5; j++)
        if((p_deal[j]>>2) != (p_deal[j-1]>>2)-1) {k &= 1; break;}
    if (k==1) rank = 5;                       //同花
    if (k==2) rank = 4;                       //顺子
    if (k==3) rank = 8;                       //同花顺
}
//记录顺子的最大值
if ((rank==4) || (rank==8))
    p_value[2] = p_deal[4]>>2;
//保存散牌或者同花的所有值
if ((rank==0) || (rank==5))
    for(j=0; j<5; j++)
        p_value[j+2] = (p_deal[j]>>2);
//一个对子时, 保存除对子外其余牌的值
if (rank==1)
{
    k = 3;
    for(j=0; j<5; j++)
        if((p_deal[j]>>2) != p_value[2])
            p_value[k++] = (p_deal[j]>>2);
}
//两个对子时, 保存除对子外其余牌的值
if(rank==2)
{

```

```

        k=4;
        for(j=0; j<5; j++)
            if((p_deal[j]>>2)!=p_value[2])
                if((p_deal[j]>>2)!=p_value[3])
                    p_value[k++] = (p_deal[j]>>2);
    }
    p_value[1] = rank;                //保存规则的级别
    p_value[0] = number[rank];        //保存牌中不同值的个数
}
//读出第一家牌中不同值的个数
int match = value[0][0];            //读出第一家不同牌的个数
int *hand1 = value[0];              //读出第一家牌的情况
int *hand2 = value[1];              //读出第二家牌的情况
//两家手中的牌比大小
while(*(++hand1) == *(++hand2))      //牌依次相同时
    if (--match<0) break;
if (match<0) printf("Tie.\n");        //牌的大小全部相同
else if (*hand1>*hand2)               //在不相同时，第一家的牌大
    printf("Black wins.\n");
else
    printf("White wins.\n");
}
return 0;
}

```

## ZJU1116-A Well-Formed Problem<sup>[1、2、3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

### Background

XML, eXtensible Markup Language, is poised to become the lingua franca of structured data communication for the foreseeable future, due in part to its strict formatting requirements. XML parsers must report anything that violates the rules of a well-formed XML document. An XML document is said to be well-formed if it meets all of the wellformedness constraints as defined by the World Wide Web Consortium (W3C) XML specification.

XML documents are composed of units called elements, that contain either character data and/or other elements.

Elements may also contain within their declaration values called attributes. Consider the

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1116>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1911>

[3] <http://www.acmgnyr.org/year2000/problems.shtml>



following XML document:

```
<?xml version="1.0"?>
<customer>
  <name>
    <first>John</first>
    <last>Doe</last>
  </name>
  <address>
    <street>
      <number>15</number>
      <direction>West</direction>
      <name>34th</name>
    </street>
    <city>New York</city>
    <state-code>NY</state-code>
    <zip-code format="PLUS4">10001-0001</zip-code>
    <country-code>USA</country-code>
  </address>
  <orders/>
</customer>
```

The bold identifiers contained within angle brackets are the elements of the document. The italicized identifier "format" within the "zip—code" element is an attribute of that element. All elements, with the exception of "orders", have a start and an end declaration, also called a tags. The "orders" element is an empty element, as indicated by the "/" sequence that closes the element, and does not require a separate end—tag. The first line is a processing instruction for an XML parser and is not considered an element of the document.

The rules for a well—formed document are:

1. There is exactly one element that is not contained within any other element. This element is identified as the "root" or "document" element. In the example above, "customer" is the document element.
2. The structure of an XML document must nest properly. An element's start—tag must be paired with a closing end—tag if it is a non—empty element.
3. The name in an element's end—tag must match the element type in the start—tag. For example, an element opened with <address> must be closed by </address>.
4. No attribute may appear more than once in the same start—tag or empty—element tag.
5. A parsed element must not contain a recursive reference to itself. For example, it is improper to include another address element within an address element.
6. A named attribute must have an associated value.

## Input

The input file will contain a series of XML documents. The start of each document is identified by a line containing only the processing instruction "<?xml version="1.0"?>". The end of the input

is identified by a line containing only the text "<?end?>" (this is not a true XML processing instruction, just a sentinel used to mark the end of the input for this problem). As with all XML documents, white space between elements and attributes should be ignored. You may make the following assumptions with regard to the input.

The only processing instruction that will be present is the XML version processing instruction, and it will always appear only at the beginning of each document in the input.

Element and attribute names are case—sensitive. For example, <Address> and <address> are considered to be different.

Element and attribute names will use only alpha—numeric characters and the dash "—" character.

XML comments will not appear in the input.

Values for attributes will always be properly enclosed in double quotes.

## Output

For each input XML document, output a line containing the text "well-formed" if the document is well-formed,"non well-formed" otherwise.

Example

## Input

```
<?xml version="1.0"?>
<acm-contest-problem>
  <title>A Well-Formed Problem</title>
  <text>XML, eXtensible Markup Language, is poised to become the lingua
franca of
structured data communication for the foreseeable future. [...]</text>
  <input>probleme.in</input>
  <output>probleme.out</output>
</acm-contest-problem>
<?xml version="1.0"?>
<shopping-list>
  <items>
    <item quantity="1" quantity="1">Gallon of milk</item>
    <item>Frozen pizza
  </items>
</Shopping-list>
<errand-list>
  <errand>Get some cash at the ATM
    <errand>Pick up dry cleaning</errand>
  </errand>
</errand-list>
<?end?>
```

## Output

```
well-formed
```

## Problem Source

Greater New York 2000

### 【题目大意】

在可预见的将来，有着严格格式要求的 XML (eXtensible Markup Language) 语言将成为数据通信的通用语言。XML 语法分析器会报告违犯标准 XML 文档的任何细节。一个规范的 XML 文档应该符合全球信息网协会 (the World Wide Web Consortium, W3C) 的规范要求。

XML 文档的组成单位称为元素 (包括字符、数据及其他元素)。

在元素中声明的值称为属性。考虑下列 XML 语言文档：

.....

在尖括号里的粗体标识符就是 XML 文档的元素。在 “zip-code” 元素中的斜体字 “format” 就是该元素的属性。除 “orders” 之外，所有元素都有起始声明和结尾声明，也称为一个标签。元素 “orders” 是空值元素，使用 “/” 结束该元素，而且不需要单独的结束标签。第一行是 XML 语法分析器的处理指令，不作为文档的一个元素。

规范的 XML 文档规则是：

(1) 只有一个元素不被其他元素所包含。这个元素就是根 “root” 或文档 “document” 元素。在上面例子中，“customer” 就是文档元素。

(2) XML 文档必须正确嵌套。如果不是空值元素，那么一个元素的起始标签必须和结束标签相匹配。

(3) 元素的结束标签名称要与开始标签名称相匹配。例如，一个元素以 <address> 开始，则必须以 </address> 结束。

(4) 在同一个开始标签及空值元素标签里，属性只能出现一次。

(5) 一个要处理的元素不能包含对自身的递归引用。例如，在一个 address 元素里不能包含另一个 address 元素。

(6) 任何声明过的属性都必须有一个值。

### 输入格式

输入多组 XML 文档。每个文档的开始是一行处理指令 “<?xml version=“1.0”?>”。输入结束标志是一行 “<?end?>” (这不是一条 XML 处理指令，只是本题中所使用的结束标记)。对于所有 XML 文档，忽略元素和属性之间的空格。假设输入如下：

(1) XML 版本处理指令只出现一次，并且它只出现在每个文档的开头。

(2) 元素和属性是大小写敏感的。例如，<Address>和<address>是不同的。

(3) 元素和属性的名称只由字母、数字以及连接符 “-” 组成。

(4) 输入文档中没有 XML 语言的注解。

(5) 属性的值，在一对双引号内。

### 输出格式

对每个 XML 文档输出一行，如果文档是规范的，那么输出 “well-formed”，否则输出 “non well-formed”。

## 【算法分析】

本题要求设计一个简单的 XML 文档的语法分析器。首先将 XML 文档中的 tag 取出来，然后根据规范进行正确性判断。

### (1) 输入 XML 文档

XML 文档的输入比较麻烦，主要是解决处理指令"<?xml version="1.0"?>"和结束标志"<?end?>"。设两个标志变量：

```
int first;           //对应第一次读取处理指令
int endFile;         //对应文档结束标志
```

然后使用 gets() 函数每次读取一行字符串，就避免了空格的干扰。

### (2) 抽取 XML 文档中的 tag

使用数组存放 tag：

```
char tag[1000][100];
```

遇到 "<" 时记录起始位置，遇到 ">" 时开始抽取 tag。但是 "</>" 这样的 tag 是非法的，因为 "/" 后面应该有名称。

表 3-3 是从样例中按顺序抽取的 tag：

表 3-3 样例 XML 文档的 tag

下标	样例 1	样例 2
0	acm-contest-problem	shopping-list
1	title	items
2	/title	item
3	text	/item
4	/text	item
5	input	/items
6	/input	/Shopping-list
7	output	errand-list
8	/output	errand
9	/acm-contest-problem	errand
10		/errand
11		/errand
12		/errand-list

结束 tag 中的 "/" 不能省略，用于配对分析。

将 tag 的个数记录到变量 count 中。如果 count 为奇数，无疑是不匹配的，直接输出否定的答案，如样例 2 (count=13)。

### (3) 根据规范进行正确性判断

由于 tag 像 for 循环一样，应该是正确嵌套的。很容易想到的方法是递归，把正确匹配的名称逐一找出来。其实没有这么复杂，从表 3-3 看出，无论怎么嵌套，最里面的名称应该直接匹配。因此不断遍历数组 tag，将相邻匹配的名称去掉，直到没有匹配项为止。

如果一个匹配项都没有，该算法会导致死循环。因此在遍历时，如果发现一个匹配项也没有，就应该停止遍历。

遍历的时候，如果发现重名的 tag，那也是非法的。

## 【程序代码】

程序名称: zju1116.c

题 目: A Well-Formed Problem

提交语言: C

运行时间: 0ms

运行内存: 160KB

---

```
#include<stdio.h>
#include<string.h>

#define N 1000
#define M 100

//判断第 i 个 tag 与第 j 个 tag 是否匹配
int pair(char tag[][M],int i,int j)
{
    if(tag[j][0]!='/') return 0;           //第 j 个 tag 应该是结束 tag
    int k;
    for(k=0; k<strlen(tag[i]); k++)        //逐个字符比较
        if(tag[i][k]!=tag[j][k+1]) break;
    if(k<strlen(tag[i])) return 0;         //tag 不匹配
    else return 1;                         //tag 匹配
}

int main()
{
    char xml[N];
    int first = 1;                          //是否第一次读取处理指令
    int endFile = 0;                        //文档结束标志
    while(1)
    {
        char tag[N][M];
        memset(tag, 0, sizeof(tag));
        int i,j;
        int count = 0;                      //tag 的个数
        while(gets(xml))
        {
            if (strcmp(xml,"<?end?>")==0) { //文档结束
                endFile = 1;
                break;
            }
            //第一次读取处理指令
            if (strcmp(xml,"<?xml version=\"1.0\"?>")==0) {
                if (first) {first = 0; continue;}
            }
        }
    }
}
```

```

        else break; //不是第一次，标志当前文档结束
    }
    //抽取 XML 文档中的 tag
    int start = -1;
    for(i=0; i<strlen(xml); i++)
    {
        if(xml[i]=='<') start = i; //起始位置
        if(xml[i]=='>' && start>=0) //结束位置
        {
            if(xml[i-1]=='/') start = -1; //非法
            else{
                //抽取 tag 名称
                char name[M];
                int k = 0;
                for(j=start+1; j<i; j++)
                {
                    if(xml[j]==' ') break;
                    name[k++] = xml[j];
                }
                name[k] = '\0';
                //追加到 tag 数组中
                strcpy(tag[count], name);
                count++;
                start = -1;
            }
        }
    }
}

int formed = 1;
if (count%2) formed = 0; //tag 的个数是奇数，非法
else if (count>0)
{
    //不断遍历数组 tag，将相邻匹配的名称去掉，直到没有匹配项为止
    while(count>0 && formed)
    {
        int noPair = 1; //无匹配项标志
        for(i=0; i<count-1 && formed; i++)
        {
            if (pair(tag,i,i+1)) //相邻的两个 tag 匹配
            {
                noPair = 0;
                //查找重名的 tag
                for(j=0; j<i && formed; j++)
                    if(strcmp(tag[j],tag[i])==0) formed = 0;
                //去掉匹配的 tag
                for(j=i; j<count-2; j++)

```

```

        strcpy(tag[j],tag[j+2]);
        count = count-2;
        i--;
    }
}
if (noPair) {formed = 0; break;}    //发现不匹配的 tag
}
if (formed) printf("well-formed\n");
else printf("non well-formed\n");
if (endFile) break;                //处理结束标志
}
return 0;
}

```

## ZJU1126-Bio-Informatics<sup>[1、2]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

Bio-informatics is an exciting new field of science, in which computer science techniques are applied to solving biological problems. The search for genetic drugs is one of the central problems of bio-informatics. In tackling this problem, genes from various organisms are compared.

A gene is characterized by the sequence of amino acids that can be derived from it.

There are altogether 20 amino acids. Each amino acid is identified by a one—letter abbreviation of its full chemical name.

(The upper case letters of the alphabet, except B, J, O, U, X, and Z, are used to identify amino acids.)

### Sample Input

human	fruitfly	nematode	yeast	bacteria
M	M	M	M	M
E	E	E	E	E
C	S	S	S	S
L	L	L	L	W
D	D	D	D	D
A	A	A	A	A
K	Q	G	N	G
C	A	A	C	K
T	T	T	T	T
S	H	E	M	R

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1126>

[2] <http://acm.uva.es/archive/nuevportal/data/problem.php?p=2081>

Each logical column of the input specifies a particular gene from a different organism. The number of organisms is at least three but not greater than eight.

The first line specifies the names of the organisms. Each name consists of at least one but not more than eight lower case characters and is right-justified in a field of width 9 characters.

Each of the remaining lines specifies an amino acid for each of the organisms listed on the first line. Each amino acid is represented by its one-letter abbreviation, right-justified in a field of width 9 characters, under the name of the organism with which it is associated. Thus, in the example shown, the amino acid sequence for the particular yeast gene is M, E, S, L, D, A, N, C, T, M.

The amino acid sequences of all organisms represented in a given input will have the same length (in this example, 10).

The minimum length of the amino acid sequences in the input is 10, the maximum length is 9999.

Each amino acid in the amino acid sequence of a particular gene occupies a certain position. The positions are numbered starting at 1 and they increase sequentially. Thus, the yeast sequence in the example shown has M in positions 1 and 10, E in position 2, A in position 6, etc.

After re-displaying the names of the organisms (in the same order as in the input), your program will look for discrepancies among the amino acid sequences of the given organisms.

## Sample Output

```
Program 7 by team X
      human fruitfly nematode   yeast bacteria
3          *
4
7          *          *          *
8          *          *          *
10         *          *          *
End of program 7 by team X
```

For those positions in which all organisms have the same amino acid (positions 1, 2, 5, 6, and 9 in the example shown) , no output will be produced.

In those positions in which not all organisms have the same amino acid (positions 3, 4 ,7, 8 and 10 in the example shown) your program will:

Print the position number.

Identify by an asterisk those organisms that deviate (in that particular position) from the most frequently occurring amino acid (in that particular position).

In position 3 of the given example, S is certainly the most frequently occurring amino acid, and human is the only organism that does not have S in position 3.

In case of a tie for the most frequently occurring amino acid in a particular position, the amino acid that has the rightmost occurrence (among those involved in the tie) will be chosen as the most frequent one.

For example, in position 8 in the given example, both C and A occur twice. We choose C as the most frequent amino acid in position 8, because its rightmost occurrence is under yeast, which is



further right than the rightmost occurrence of A (nematode) in this position.

In the given example, there is an extreme case of a tie in position 10: all five organisms have different amino acids. Therefore, the amino having the rightmost occurrence, namely R, will be designated as the most frequent one.

Two lines of the above output are reproduced here with a formatting template:

```

      1      2      3      4      5      6      7
123456789012345678901234567890123456789012345678901234567890
      human fruitfly nematode   yeast bacteria
      3      *
```

Note in particular:

The position numbers are right justified in columns 1 — 4.

Column 5 is blank.

Starting in column 6, the output (name of an organism or an asterisk) will be right—justified in fields of width 9.

Also pay attention to formatting details, such as upper/lower case variations, blank spaces, and the absence of blank lines.

## Problem Source

Rocky Mountain 2000

### 【题目大意】

生物信息学是一门令人兴奋的科学新领域，借助计算机科学技术，解决生物信息学问题。搜寻遗传基因药物就是生物信息学的主要问题之一。在解决问题时，要比较各种不同的生物基因。

氨基酸序列能反映基因的结构。一共有 20 种氨基酸。每种氨基酸由其化学名称的一个简写字母表示。（除了 B、J、O、U、X 和 Z 以外的大写字母，都用来表示氨基酸。）

### 输入格式

输入中每列代表不同有机体的基因。有机体最少三种，最多 8 种。

第一行是有机体的名称。每个名称由至少一个，至多 8 个小写字符表示，且宽度为 9，右对齐。

剩下的每行，是第一行每个有机体的一种氨基酸。每个氨基酸的名称是一个缩写字母，右对齐，宽度为 9，位置在相应有机体的名称下面。上例中，某个酵母基因的氨基酸序列是 M、E、S、L、D、A、N、C、T、M。

输入中，所有有机体的氨基酸序列长度是相同的（本例中是 10）。

输入的氨基酸序列最小长度是 10，最大长度是 9999。

每个基因氨基酸序列的氨基酸都有一个特定的位置。位置从 1 开始按升序编号。本例中，酵母氨基酸序列中位置 1 和 10 是 M，位置 2 是 E，位置 6 是 A，等等。

与输入相同的顺序，输出有机体名字。

编程任务：找出给定生物氨基酸序列之间的差异。

### 输出格式

如果所有的有机体含相同的氨基酸（如本例中的 1，2，5，6 和 9 行），其相应位置不

必输出。

如果有机体的氨基酸互不相同（如本例中的 3, 4, 7, 8 和 10），在相应位置，你的程序将输出位置序号。

将有机体的氨基酸出现频率最少的位置标示星号。

上例中，在 3 号位置这一行，S 是出现频率最多的氨基酸，而人类（human）的基因中没有 S。

在某个位置，如果出现频率最多的氨基酸不止一个，（在出现频率相同的氨基酸中）选择最右边的那个氨基酸为出现频率最多的氨基酸。

例如，例子中的位置 8，C 和 A 都出现两次。我们选择 C 作为 8 号位置频率出现最多的氨基酸，因为在酵母（yeast）这一栏，它是最右边的，比 A（nematode）的位置还要靠右。

上例中，位置 10 是一个极端的例子：所有 5 种有机体的氨基酸都不同。因此，将出现在最右边的 R，选为出现频率最多的氨基酸。

.....

特别注意：

1~4 列中的位置序号是右对齐的。

第 5 列是空格。

从第 6 列开始，输出（有机体名称或星号）右对齐且宽度为 9。

注意格式细节，像大写/小写变化、空格，出现或缺少的空行。

## 【算法分析】

本题是字符处理。在一行中有很多不同的字符，找到出现频率最高的字符；如果有多个字符出现的频率都是最高的，就取最右边的字符。如果所有的字符都相同，就不必输出；否则，与出现频率最高的字符不一样的字符，对应位置输出星号。

### （1）数据结构

输入/输出数据的每一行字符串：

```
char line[128];
```

统计每个字符出现的频率：

```
int freq[26];
```

实际的字符：

```
char amino[21];
```

字符的个数：

```
int n;
```

出现频率最高的字符：

```
char most;
```

最右边的位置：

```
int rightmost;
```

### （2）找到出现频率最高的字符和最右边的位置

统计每个字符出现的频率，根据频率就可以找到出现频率最高的字符。最大值比较时，使用“ $\geq$ ”，得到的就是最右边的位置。

### （3）构造格式化输出

使用函数 `strcat()`，把要输出的一行构造出来，输出就方便了。

## 【程序代码】

程序名称: zju1126.c  
题 目: Bio-Informatics  
提交语言: C  
运行时间: 0ms  
运行内存: 160KB

```
#include <stdio.h>
#include <string.h>

int main()
{
    int i, j;
    int n;                                //字符的个数
    int iCase = 0;
    char line[128];                        //输入/输出数据的一行字符串
    int freq[26];                          //存放每个字符出现的频率
    char amino[21];                        //存放实际的字符
    char most;                             //出现频率最高的字符

    gets(line);
    printf("Program 7 by team X\n");
    printf("    %s\n", line);
    n = strlen(line) / 9;
    while (gets(line))
    {
        iCase++;
        memset(freq, 0, sizeof(freq));
        int rightmost = 0;                 //最右边的位置
        //对每个字符处理
        j = 0;
        for (i = 0; i < n; i++)
        {
            while(line[j++] == ' ');
            amino[i] = line[j-1];          //得到有效的字符
            //统计每个字符出现的频率
            freq[amino[i]-'A']++;
            //找到出现频率最高的字符和最右边的位置
            if (freq[amino[i]-'A'] >= rightmost)
            {
                rightmost = freq[amino[i]-'A'];
                most = amino[i];
            }
        }
        if (rightmost == n) continue;      //所有字符相同
        printf("%4d ", iCase);
```

```

memset(line, 0, sizeof(line));
//构造格式化输出
for (i = 0; i < n; i++)
{
    strcat(line, "      ");
    if (amino[i] == most) strcat(line, " ");
    else strcat(line, "*");
}
//去掉最右端的空格
j= strlen(line)-1;
while (line[j] == ' ') j--;
line[++j] = '\0';
printf("%s\n", line);           //输出结果
}
printf("End of program 7 by team X\n");
return 0;
}

```

ZJU1159-487-3279<sup>[1、2、3]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768KB

---

Businesses like to have memorable telephone numbers. One way to make a telephone number memorable is to have it spell a memorable word or phrase. For example, you can call the University of Waterloo by dialing the memorable TUT-GLOP. Sometimes only part of the number is used to spell a word. When you get back to your hotel tonight you can order a pizza from Gino's by dialing 310-GINO. Another way to make a telephone number memorable is to group the digits in a memorable way. You could order your pizza from Pizza Hut by calling their “three tens” number 3-10-10-10.

The standard form of a telephone number is seven decimal digits with a hyphen between the third and fourth digits (e.g. 888-1200). The keypad of a phone supplies the mapping of letters to numbers, as follows:

```

A, B, and C map to 2
D, E, and F map to 3
G, H, and I map to 4
J, K, and L map to 5
M, N, and O map to 6
P, R, and S map to 7
T, U, and V map to 8
W, X, and Y map to 9

```

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1159>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1002>

[3] <http://online-judge.uva.es/p/v7/755.html>

There is no mapping for Q or Z. Hyphens are not dialed, and can be added and removed as necessary. The standard form of TUT-GLOP is 888-4567, the standard form of 310-GINO is 310-4466, and the standard form of 3-10-10-10 is 310-1010.

Two telephone numbers are equivalent if they have the same standard form. (They dial the same number.)

Your company is compiling a directory of telephone numbers from local businesses. As part of the quality control process you want to check that no two (or more) businesses in the directory have the same telephone number.

This problem contains multiple test cases!

The first line of a multiple input is an integer  $N$ , then a blank line followed by  $N$  input blocks. Each input block is in the format indicated in the problem description. There is a blank line between input blocks.

The output format consists of  $N$  output blocks. There is a blank line between output blocks.

## Input

The input will consist of one case. The first line of the input specifies the number of telephone numbers in the directory (up to 100000) as a positive integer alone on the line. The remaining lines list the telephone numbers in the directory, with each number alone on a line. Each telephone number consists of a string composed of decimal digits, uppercase letters (excluding Q and Z) and hyphens. Exactly seven of the characters in the string will be digits or letters.

## Output

Generate a line of output for each telephone number that appears more than once in any form. The line should give the telephone number in standard form, followed by a space, followed by the number of times the telephone number appears in the directory. Arrange the output lines by telephone number in ascending lexicographical order. If there are no duplicates in the input print the line:

No duplicates.

## Sample Input

```
1
12
4873279
ITS-EASY
888-4567
3-10-10-10
888-GLOP
TUT-GLOP
967-11-11
310-GINO
F101010
888-1200
```

-4-8-7-3-2-7-9-  
487-3279

## Sample Output

310-1010 2  
487-3279 4  
888-4567 3

## Problem Source

East Central North America 1999

### 【题目大意】

商人喜欢使用容易记住的电话号码。有一个办法使电话号码容易记住：用一个好记的单词或词组来表达。例如：你拨打容易记的 TUT-GLOP，就可以打电话到 Waterloo 大学。有时，数字也可以成为拼写的一部分。当你晚上回到旅馆时，想从 Gino 比萨店叫一份比萨的时候，就可以拨打 310-GINO。另一个使电话号码容易记住的方法是，把电话号码按容易记忆的方式分组。如果你想叫一份 Pizza Hut 比萨店的比萨，就会想起他们的电话号码是“3 个 10”，即 3-10-10-10。

一个标准的电话号码格式为：7 个十进制数字，且在第三及第四个数字之间有一个连字符（例如：888-1200）。以下就是电话键盘上英文字母所在数字键的位置：

A, B, C 在按键 2  
D, E, F 在按键 3  
G, H, I 在按键 4  
J, K, L 在按键 5  
M, N, O 在按键 6  
P, R, S 在按键 7  
T, U, V 在按键 8  
W, X, Y 在按键 9

按键上没有 Q 和 Z。拨号时连字符是不需要拨的，可以根据需要加在电话号码中或者从电话号码中去掉。所以 TUT-GLOP 的标准格式为：888-4567，310-GINO 的标准格式为：310-4466，3-10-10-10 的标准格式为：310-1010。

两个电话号码，如果他们的标准格式是相同的，则这两个电话号码就是同一个号码（因为都拨打相同的键）。

你的公司根据当地商人的电话号码，正在编辑电话号码簿。作为质量控制过程的一个环节，你需要检查电话号码簿中是否有两个（或者多个）商人有相同的电话号码。

本题包含多组测试例！

多组测试例的第一行是一个整数  $N$ ，然后是一个空行，接着是  $N$  个输入数据块。每个数据块的格式在问题描述中给出。每个数据块之间有一个空行。

输出格式包括  $N$  个输出数据块，每个输出数据块之间有一个空行。

### 输入格式

输入只有一个测试例（指每个数据块内）。输入的第一行，是电话号码簿中电话号码的数量（多至 100000 个），是一个正整数。接下来的各行就是电话号码簿中电话号码的列表，每个

电话号码一行，是由数字或大写英文字母（Q 和 Z 除外）以及连字符组成。字符串中的 7 个字符，全部是数字和英文字母。

### 输出格式

当一个电话号码，不管以什么方式出现超过一次时，就输出一行：转换成标准格式的电话号码，一个空格，该电话号码在电话号码簿中出现的次数。电话号码按升序排列。如果没有任何电话号码是重复的，请输出：

No duplicates.

### 【算法分析】

将字母、数字和连字符构成的电话号码，转换成标准格式的电话号码：三个数字，一个连字符和 4 个数字，再把重复的电话号码统计出来。

#### （1）数据结构

手机按键上字母与数字的映射表：

```
char map[26];
```

电话号码簿中的全部电话号码：

```
char phone[100000][9];
```

电话号码簿中电话号码的数量：

```
int n;
```

（2）读取电话号码簿中的全部电话号码，存放到数组 `phone` 中一个电话号码：

```
char number[30];
```

长度小于 20 会 “Wrong Answer”，说明电话号码中，有许多连字符。

将原来电话号码中所有的连字符全部去掉，在第三个位置加上一个连字符即可。而字母键使用数组 `map` 进行映射。

#### （3）全部电话号码排序

目的是找到相同的电话号码（集中在一起）和按升序输出。

#### （4）统计重复的电话号码，并输出结果

电话号码重复的次数：

```
int count;
```

电话号码重复的标志：

```
int flag;
```

对全部电话号码顺序搜索一遍，即可完成任务。搜索完毕后，对最后一个电话号码也要判断一下是否是重复的。

### 【程序代码】

---

程序名称： zju1159.c

题 目： 487-3279

提交语言： C

运行时间： 1540ms

运行内存： 1916KB

---

```
#include <stdio.h>
#include <stdlib.h>
```

```

#include <memory.h>
//手机按键上字母与数字的映射表
char map[26] = {'2', '2', '2', '3', '3', '3', '4', '4', '4',
               '5', '5', '5', '6', '6', '6', '7', 0, '7', '7',
               '8', '8', '8', '9', '9', '9', 0};

char phone[100000][9];           //电话号码簿中的全部电话号码
int n;                           //电话号码簿中电话号码的数量
char number[30];                 //一个电话号码

int main() {
    int i, j;

    int N;                       //数据块的数量
    scanf("%d", &N);
    while (N--)
    {
        int pos;
        //读取电话号码簿中的全部电话号码，存放到数组 phone 中
        memset(phone, 0, sizeof(phone));
        scanf("%d", &n);
        for(i = 0; i < n; i++)
        {
            scanf("%s", number );
            pos = 0;
            for(j = 0; number[j]; j++)
            {
                //去掉连字符
                if (number[j] == '-' ) continue;
                //将字母映射成相应按键的数字
                if (number[j] >= 'A' && number[j] <= 'Z' )
                    number[j] = map[number[j]-'A'];
                phone[i][pos++] = number[j];
                //第 3 个位置是标准的连字符位置
                if(pos == 3) phone[i][pos++] = '-';
            }
        }
        //全部电话号码排序，以便找到相同的电话号码和按升序输出
        qsort( phone, n, 9, (int (*)(const void *, const void *))strcmp );
        //找到相同的电话号码和按升序输出
        int count = 1;           //重复的次数
        int flag = 0;            //重复的标志
        for( i = 1; i < n; i++)
            //相邻的两个电话号码比较
            if(strcmp(phone[i-1], phone[i]))
            {

```



```

        //输出前一个有重复的电话号码
        if( count > 1 ) {
            printf( "%s %d\n", phone[i-1], count );
            flag = 1;
        }
        count = 1;
    }
    else count++;
    //相邻的两个电话号码相同时
    //最后一个电话号码也是重复的
    if( count > 1 ) {
        printf( "%s %d\n", phone[i-1], count );
        flag = 1;
    }
    //一个重复的电话号码也没有
    if( !flag ) printf( "No duplicates.\n" );
    if (N) printf("\n");
}
return 0;
}

```

## 第四章 基本数据结构题

### ZJU1094-Matrix Chain Multiplication<sup>[1、2、3]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768K

---

Matrix multiplication problem is a typical example of dynamical programming.

Suppose you have to evaluate an expression like  $A*B*C*D*E$  where  $A, B, C, D$  and  $E$  are matrices. Since matrix multiplication is associative, the order in which multiplications are performed is arbitrary. However, the number of elementary multiplications needed strongly depends on the evaluation order you choose.

For example, let  $A$  be a  $50*10$  matrix,  $B$  a  $10*20$  matrix and  $C$  a  $20*5$  matrix.

There are two different strategies to compute  $A*B*C$ , namely  $(A*B)*C$  and  $A*(B*C)$ .

The first one takes 15000 elementary multiplications, but the second one only 3500.

Your job is to write a program that determines the number of elementary multiplications needed for a given evaluation strategy.

#### Input Specification

Input consists of two parts: a list of matrices and a list of expressions.

The first line of the input file contains one integer  $n(1 \leq n \leq 26)$ , representing the number of matrices in the first part. The next  $n$  lines each contain one capital letter, specifying the name of the matrix, and two integers, specifying the number of rows and columns of the matrix.

The second part of the input file strictly adheres to the following syntax (given in EBNF):

```
SecondPart = Line { Line } <EOF>
Line       = Expression <CR>
Expression = Matrix | "(" Expression Expression ")"
Matrix     = "A" | "B" | "C" | ... | "X" | "Y" | "Z"
```

#### Output Specification

For each expression found in the second part of the input file, print one line containing the word "error" if evaluation of the expression leads to an error due to non-matching matrices. Otherwise print one line containing the number of elementary multiplications needed to evaluate the expression in the way specified by the parentheses.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1094>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2246>

[3] <http://acm.uva.es/p/v4/442.html>

# Sample Input

9	A
A 50 10	B
B 10 20	C
C 20 5	(AA)
D 30 35	(AB)
E 35 15	(AC)
F 15 5	(A (BC) )
G 5 10	(( (AB) C)
H 10 20	(( (( (DE) F) G) H) I)
I 20 25	(D (E (F (G (HI) ) ) ) )
	(( (D (EF) ) ) ((GH) I) )

# Sample Output

0  
0  
0  
error  
10000  
error  
3500  
15000  
40500  
47500  
15125

# Problem Source

University of Ulm Local Contest 1996

## 【题目大意】

矩阵乘法问题是动态规划的一个典型例子。

假设你要计算  $A \times B \times C \times D \times E$ ，其中  $A$ 、 $B$ 、 $C$ 、 $D$  和  $E$  是矩阵。由于矩阵相乘具有结合性，所以矩阵相乘的顺序是任意的。但是，矩阵相乘时做乘法的次数，取决于你所选择的矩阵相乘的顺序。

例如，矩阵  $A$ ：  $50 \times 10$ ，  $B$ ：  $10 \times 20$ ，  $C$ ：  $20 \times 5$ 。

计算  $A \times B \times C$  有两种顺序，即  $(A \times B) \times C$  和  $A \times (B \times C)$ 。

第一种顺序需要做乘法 15000 次，而第二种只要 3500 次。

编程任务：对给定的矩阵相乘顺序，计算矩阵相乘时所需的乘法次数。

## 输入格式

输入分为两部分：矩阵列表和矩阵相乘的表达式列表。

第一行是一个整数  $n$  ( $1 \leq n \leq 26$ )，表示矩阵数目。接着有  $n$  行，每行的开头是一个大写字母，是矩阵的名称，然后是两个整数，表示该矩阵的行数与列数。

输入的第二部分严格遵守下列语法（用 EBNF 表示）：

.....

### 输出格式

对第二部分的每个表达式，输出一行：根据表达式中矩阵相乘的顺序，如果相乘时矩阵不匹配，输出“error”，否则输出矩阵相乘时做乘法的次数。

### 【算法分析】

本题是根据给定的矩阵相乘的顺序，计算矩阵相乘时做乘法的次数。在University of Ulm Local Contest 1996 的竞赛资料页面<sup>[1]</sup>上，给出了一个标程。主要是采用递归的方法模拟矩阵相乘的过程。

对给定的矩阵顺序表达式：

```
char e[100];
```

分为两种情况：

①遇到左括号“(”

表示遇到了这样的情况：(expr1, expr2)

其中表达式 expr1 和 expr2, 其值是矩阵名称 *t1* 和 *t2*; 并跳过右括号。如果 expr1 和/或 expr2 不是矩阵名称，递归计算以后肯定能够得到矩阵名称。计算矩阵 *t1* 和 *t2* 相乘时的乘法次数。

②如果不是左括号“(”，那就是矩阵名称，返回该名称，这可能就是 *t1* 或 *t2*。

### 【程序代码】

---

程序名称：	zju1094.c
题目：	Matrix Chain Multiplication
提交语言：	C
运行时间：	00:00.00
运行内存：	388K

---

```
#include <stdio.h>

//矩阵的数据结构
typedef struct {
    int mults;           //得到该矩阵时，已经做过乘法的次数
    int rows;           //行数
    int cols;           //列数
}Node;

int rows[26],cols[26];  //给定矩阵的行数和列数
char e[100];           //计算顺序的表达式
int pos;               //在表达式 e 中的位置
char error;            //失败标志

//递归计算矩阵相乘
Node expression()
{
```

---

[1] <http://www.informatik.uni-ulm.de/acm/Locals/1996/>

```

Node t;
//遇到左括号'(', 表示这样的情况: (expr1, expr2)
if (e[pos]=='(')
{
    Node t1,t2;
    pos++;
    t1 = expression();           //获得表达式 expr1
    t2 = expression();           //获得表达式 expr2
    pos++;                       //跳过右括号
    if (t1.cols != t2.rows) error = 1; //两个矩阵不能相乘
    //计算矩阵 t1 和 t2 相乘时, 做乘法的次数
    t.rows = t1.rows;
    t.cols = t2.cols;
    t.mults = t1.mults + t2.mults + t1.rows * t1.cols * t2.cols;
}
else
{
    //肯定是一个矩阵
    t.rows = rows[e[pos]-'A'];    //矩阵的行数
    t.cols = cols[e[pos]-'A'];    //矩阵的列数
    t.mults = 0;
    pos++;                       //下一个位置
}
return t;
}

int main()
{
    int i;
    int n;                       //矩阵的个数
    char name;                   //矩阵的名称
    int row, col;
    Node t;
    //读取数据的第一部分
    scanf("%d\n", &n);
    for (i=0; i<n; i++)
    {
        scanf("%c%d%d\n", &name, &row, &col);
        rows[name-'A'] = row;
        cols[name-'A'] = col;
    }
    //对每一个表达式进行计算
    while (gets(e))
    {
        pos = error = 0;
        t = expression();
        if (error) puts("error"); //矩阵不能相乘
    }
}

```

```

        else printf("%d\n", t.mults);           //输出做乘法的次数
    }
    return 0;
}

```

## 【其他算法】

在上面的算法和编码中，花费了大量的精力处理字符串，这在竞赛中是很浪费时间的。利用 C++ 强大的标准模板库 STL，可以使编码更简单，节约很多时间。

用 map() 保存矩阵参数：

```
map<char, Node> matrix;
```

其中 char 是矩阵名称的数据类型，Node 是矩阵行列数的数据结构。

用 stack() 模拟矩阵的乘法：

```
stack<Node> array;
```

计算方法很简单，遇到矩阵就进栈；遇到右括号就将栈顶的两个矩阵相乘，并将结果再压入栈中。

## 【代码程序】

---

程序名称：	zju1094-map-stack.cpp
题    目：	Matrix Chain Multiplication
提交语言：	C++
运行时间：	00:00.00
运行内存：	840K

---

```

#include <map>
#include <stack>
#include <iostream>
using namespace std;

struct Node { int row, col; };           //矩阵的行列数

int main()
{
    int n;                               //矩阵的个数
    char name;                           //矩阵的名称
    map<char, Node> matrix;              //矩阵参数
    //读取数据的第一部分
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cin >> name;
        cin >> matrix[name].row >> matrix[name].col;
    }
    string exp;
    //对每一个表达式进行计算
    while(cin >> exp)

```

```

{
    int i;
    int count = 0;                                //矩阵做乘法的次数
    stack<Node> array;                             //模拟矩阵的乘法
    //对表达式的每一个字符
    for(i = 0; i < exp.size(); i++)
    {
        if(exp[i] == '(') continue;                //左括号
        //遇到右括号时，将栈顶两个矩阵相乘，再压入堆栈
        if(exp[i] == ')')
        {
            Node b = array.top();
            array.pop();
            Node a = array.top();
            array.pop();
            if(a.col != b.row)                      //两个矩阵不能相乘
            {
                cout << "error" << endl;
                break;
            }
            //累计两个矩阵相乘的次数
            count += a.row * b.row * b.col;
            //将计算得到的新矩阵入栈
            Node tmp = {a.row, b.col};
            array.push(tmp);
        }
        else array.push(matrix[exp[i]]);           //矩阵入栈
    }
    if(i == exp.size())                            //输出结果
        cout << count << endl;
}
return 0;
}

```

## ZJU1097-Code the Tree<sup>[1、2]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768K

---

A tree (i.e. a connected graph without cycles) with vertices numbered by the integers 1, 2, ...,  $n$  is given. The "Prufer" code of such a tree is built as follows: the leaf (a vertex that is incident to only

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1097>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2567>

one edge) with the minimal number is taken. This leaf, together with its incident edge is removed from the graph, while the number of the vertex that was adjacent to the leaf is written down. In the obtained graph, this procedure is repeated, until there is only one vertex left (which, by the way, always has number  $n$ ). The written down sequence of  $n-1$  numbers is called the Prufer code of the tree.

Your task is, given a tree, to compute its Prufer code. The tree is denoted by a word of the language specified by the following grammar:

```
T ::= "(" N S ")"
S ::= " " T S
    | empty
N ::= number
```

That is, trees have parentheses around them, and a number denoting the identifier of the root vertex, followed by arbitrarily many (maybe none) subtrees separated by a single space character. As an example, take a look at the tree in the figure below which is denoted in the first line of the sample input.

Note that, according to the definition given above, the root of a tree may be a leaf as well. It is only for the ease of denotation that we designate some vertex to be the root. Usually, what we are dealing here with is called an "unrooted tree".

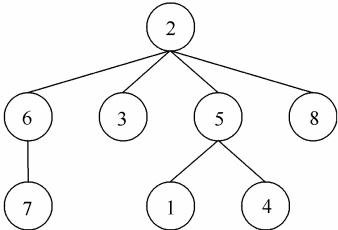


Figure 4-1

### Input Specification

The input contains several test cases. Each test case specifies a tree as described above on one line of the input file. Input is terminated by EOF. You may assume that  $1 \leq n \leq 50$ .

### Output Specification

For each test case generate a single line containing the Prufer code of the specified tree. Separate numbers by a single space. Do not print any spaces at the end of the line.

### Sample Input

```
(2 (6 (7)) (3) (5 (1) (4)) (8))
(1 (2 (3)))
(6 (1 (4)) (2 (3) (5)))
```

### Sample Output

```
5 2 5 2 6 2 8
2 3
2 1 6 2 6
```

### Problem Source

University of Waterloo Local Contest 2001.06.02

#### 【题目大意】

树（即无环图）的顶点，用整数  $1, 2, \dots, n$  编号。Prufer 码是按如下步骤构造的树：找到编



号最小的叶结点（只有一条边与该顶点相连）。将该叶结点，及其相连的那条边，从图中删掉。同时，记下与它连接的那个结点的编号。重复上面的步骤，直到剩下最后一个结点（顺便说一下，这个数就是  $n$ ）。写下来的  $n-1$  个数的序列，就是该树的 **Prufer 码**。

**编程任务：**根据输入的树，计算该树的 **Prufer 码**。描述树的语法规则如下：

.....

也就是，树的周围有括号。第一个数是根结点编号，后面跟有任意个子树（也可能一个也没有），中间有一个空格。例如，图 4-1 所示的树，就是输入样例中的第一行。

注意：根据上面给出的描述，树的根结点也可能就是一个叶结点。为便于说明，我们指定一些结点作为根结点。通常，我们将正在处理的树称为“**unrooted tree**”。

**输入格式**

输入包含多组测试例。每个测试例一行，按上面的规则描述一棵树。遇到 EOF 输入结束。 $1 \leq n \leq 50$ 。

**输出格式**

对每组测试例，输出一行，是该树的 **Prufer 码**。数字之间用空隔分开，行末没有空格。

**【算法分析】**

**（1）样例分析**

对第一样例，就是题目中的图。首先找到编号最小的叶结点，编号是 1；将该叶结点及所在边去掉，并记下与之相连的结点编号：5。如图 4-2 所示。

接着找编号最小的叶结点，编号是 3；将该叶结点及所在边去掉，并记下与之相连的结点编号：2。如图 4-3 所示。

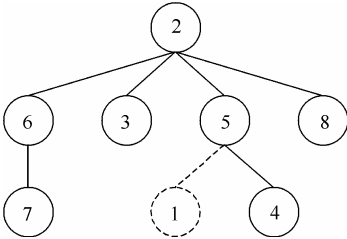


图 4-2 去掉第一个叶结点 1

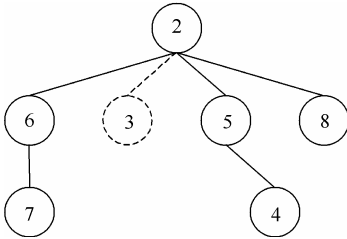


图 4-3 去掉第二个叶结点 3

这样继续下去，就得到答案了。

本题的算法和代码参考自University of Waterloo Local Contest的竞赛网站<sup>[1]</sup>。

**（2）数据结构**

将每个结点的相邻结点建立对应关系，如表 4-1 所示：

表 4-1 结点的相邻结点

结点编号	1	2	3	4	5	6	7	8
相邻结点	5	3 5 6 8	2	5	1 2 4	2 7	6	2

使用 C++ 标准模板库 STL 的 **Vector()** 和 **Set()** 很容易实现表 4-1 的数据结构：

[1] <http://plg1.cs.uwaterloo.ca/~acm00/>

```
vector<set<int> > adj (1024, set<int>());
```

输入数据与结点编号的转换由函数 `parse()` 实现:

```
void parse (vector<set<int> > &adj, unsigned int p = 0)
```

初值 `p=0` 表示从一行的左边第一次读取数据, 才读取结点编号, 还没有相邻结点编号。

对一个左括号, 无论里面有什么样的数据, 它总会遇到与其对应的右括号, 这正好是一个递归的思想。

读取数据时, 由于字符之间存在空格, 采用 `cin>>ws`, 就可以跳过空格。变量 `ws` 不用定义, 它是标准输入流中的控制符。

### (3) 实现 Prufer 编码

从表 4-1 可以看出, 当某结点只有一个相邻结点时, 它就是叶子结点。将所有这些叶子结点放到一个优先队列中, 选出结点编号最小的那个结点, 输出其相邻结点的编号, 就是该结点的 Prufer 编码。对于表 4-1 的样例数据, 初始状态的叶子结点如下:

1, 3, 4, 7, 8

将这些结点放到优先队列中自动排序, 就获得了编号最小的叶子结点 1, 输出其相邻结点的编号 5, 同时将结点 5 相邻结点中的结点 1 删除。如果结点 5 只有一个结点, 则进入优先队列。

优先队列:

```
priority_queue< int, vector<int>, greater<int> > leafs;
```

其中 `greater<int>` 是队列的排序因子。

## 【程序代码】

---

程序名称:	<code>zju1097.cpp</code>
题 目:	<code>Code the Tree</code>
提交语言:	<code>C++</code>
运行时间:	<code>00:00.04</code>
运行内存:	<code>880K</code>

---

```
#include <iostream>
#include <queue>
#include <set>
#include <vector>
using namespace std;

//读取数据, 实现结点的数据结构
void parse (vector<set<int> > &adj, unsigned int p = 0)
{
    unsigned int x;
    //读取结点编号
    cin >> ws >> x;
    //p≠0 时, 表示 p 是前面读取的结点编号
    if (p)
    {
        //结点的相邻是对称的
        adj[p].insert (x);
        adj[x].insert (p);
    }
}
```

```

while (true)
{
    char ch;
    //读取括号
    cin >> ws >> ch;
    //如果是')', 递归返回
    if (ch == ')') break;
    //否则是'(', 递归调用
    parse (adj, x);
}
return;
}

int main ()
{
    char ch;
    while (cin >> ws >> ch)
    {
        //构造结点数据结构 adj
        vector<set<int> > adj (1024, set<int>());
        parse (adj);
        //定义优先队列 leafs
        priority_queue< int, vector<int>, greater<int> > leafs;
        //结点的个数
        int n = 0;
        //对每一个结点, 判断其相邻结点的个数是否是 1
        for (unsigned int i=0; i<adj.size(); i++)
            if (adj[i].size())
            {
                n++; //统计结点的个数
                if (adj[i].size() == 1) //是叶子结点
                    leafs.push (i); //进入优先队列
            }
        //处理优先队列
        for (int k=1; k<n; k++)
        {
            unsigned int x = leafs.top(); //第一个结点 x
            leafs.pop();
            //结点 x 的相邻结点是 p。
            //注意: 数组 adj 的元素是集合, 获得集合的元素要用指针
            unsigned int p = *(adj[x].begin());
            if (k > 1)
                cout << " ";
            cout << p;
            //对称的, 在结点 p 中删除其相邻结点 x
            adj[p].erase(x);
        }
    }
}

```

```

//如果结点 p 也成为叶子结点，则进入优先队列
if (adj[p].size() == 1)
    leafs.push (p);
}
cout << endl;
}
return 0;
}

```

本题的 C 语言实现要麻烦一点，感兴趣的读者可以参考光盘中的代码：zju1097.c。

## ZJU1156-Unscrambling Images<sup>[1、2、3]</sup>

Time Limit: 1 Second

Memory Limit: 32768KB

Quadtrees are commonly used for encoding digital images in a compact form. Given an  $n*n$  image (where  $n$  is a power of 2,  $1 \leq n \leq 16$ ), its quadtree encoding is computed as follows. Start with a quadtree with exactly one node, namely the root, and associate with this node the  $n*n$  square region for the entire image. Then the following is performed recursively:

If every pixel in the region associated with the current node has an intensity value of  $p$ , then the node is made a leaf and it is assigned an associated value of  $p$ .

Otherwise, four nodes are added as children of the current node. The region is divided into four equal (square) quadrants and each quadrant is assigned to one child node. The algorithm recurses on each of the children nodes.

When the process terminates, we obtain a quadtree in which every internal node has four children. Every leaf node has an associated value representing the intensity of the region corresponding to the leaf node. An example of an image and its quadtree encoding is shown below.

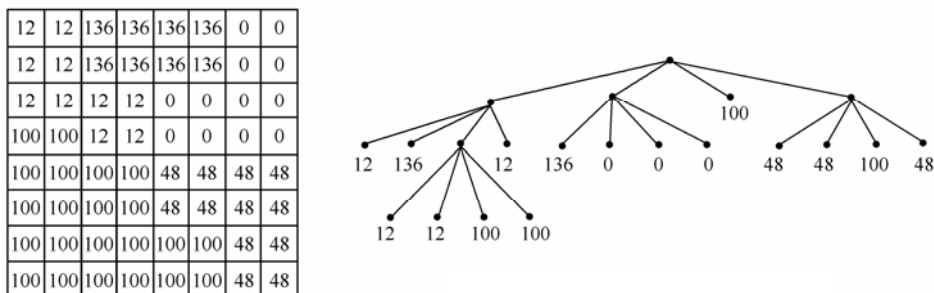


Figure 4-4

We have assumed that the four children represent, from left to right, the upper left, upper right, lower left, and lower right quadrants, respectively.

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1156>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1086>

[3] <http://online-judge.uva.es/p/v7/752.html>

To easily identify a node in a quadtree, we assign a number to each node by the following rules:  
The root is numbered 0.

If the number of a node is  $k$ , then its children, from left to right, are numbered  $4k+1$ ,  $4k+2$ ,  $4k+3$ ,  $4k+4$ .

Images encoded as quadtrees can be encrypted by a secret password as follows: whenever a subdivision is performed, the four branches are reordered. The reordering may be different at each node, but it is completely determined by the password and the node number.

Unfortunately, some people use the “save password” feature in the encoding program and use the same password for multiple images. By observing the encoding of a well-chosen test image, any image encoded with the same password can be decoded without the password. In this test image, each pixel has a distinct intensity from 0 to  $n^2-1$  arranged from left-to-right, top-to-bottom in increasing order. An example for  $n=16$  is given below:

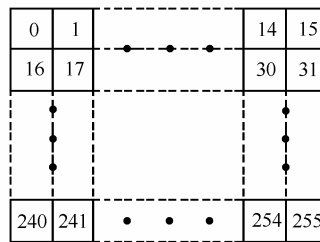


Figure 4-5

You managed to gain access to the encoding program and used it to encode the test image. Given the quadtree encoding of the test image, write a program to decode any other image encoded with the same password.

This problem contains multiple test cases!

The first line of a multiple input is an integer  $N$ , then a blank line followed by  $N$  input blocks. Each input block is in the format indicated in the problem description. There is a blank line between input blocks.

The output format consists of  $N$  output blocks. There is a blank line between output blocks.

## Input

You will be given a number of cases in the input. The first line of input consists of a positive integer indicating the number of test cases to follow. Each test case starts with a line containing  $n$ , followed by the quadtree encoding of the test image and the quadtree encoding of the secret image to be decoded. Each quadtree encoding starts with a line containing a positive integer  $m$  indicating the number of leaf nodes in the tree. The next  $m$  lines are of the form:

$k$  intensity

which specifies that the node numbered  $k$  is a leaf node with the specified intensity as the associated leaf value. Nodes not specified are either internal nodes or absent in the quadtree. You may assume that all intensities are between 0 and 255, inclusively. You may also assume that each quadtree encoding is a valid output obtained from the encoding algorithm described above.

# Output

For each test case, print the case number followed by a blank line. Then, print the intensities of the pixels of the decoded image one row at a time. Print each intensity right-justified in a field of width 4, with no extra spaces between fields. Insert a blank line between cases.

# Sample Input

1	2	4	4	7
2	4	1 23	16	2 10
	1 3	2 123	5 8	3 20
	2 2	3 253	6 9	4 30
	3 0	4 40	7 13	5 41
	4 1		8 12	6 42
			9 0	7 44
			10 4	8 43
			11 1	
			12 5	
			13 2	
			14 3	
			15 7	
			16 6	
			17 10	
			18 11	
			19 15	
			20 14	

# Sample Output

Case 1
253 40
123 23
Case 2
10 10 20 20
10 10 20 20
41 42 30 30
43 44 30 30

# Problem Source

East Central North America 1999; Pacific Northwest 1999

## 【题目大意】

四叉树通常是以紧凑的形式对数字图像进行编码。已知一个  $n \times n$  的图像（其中  $n$  是 2 的幂， $1 \leq n \leq 16$ ），其四叉树编码的计算如下。从只有一个结点的四叉树（即根结点）开始，然后将整个图像的  $n \times n$  正方形矩阵与该结点关联在一起。下面进行递归：

- 如果与当前结点相关联的区域中，每个像素都是相同的色饱和度  $p$  值，则该结点作为

一个叶子结点，相关联的值是  $p$ 。

- 否则，为当前结点增加 4 个子结点。相关联的区域分为 4 个相等的（正方形）象限，给每个象限分配一个子结点。对每个子结点递归地应用该算法。

当处理过程结束时，我们得到了一棵四叉树，其每个内部结点都有 4 个子结点。每个叶子结点都有一个相关联的值，表示与该叶结点相关联区域的色饱和度。如图 4-5 所示，是一个示例图像和相应的四叉树编码。

我们假定 4 个子结点按从左至右的顺序，分别表示相关联区域的左上角，右上角，左下角和右下角的象限。

为了容易地确定四叉树的一个结点，我们按以下规则给每个结点编号：

- 根编号为 0。
- 如果结点的编号为  $k$ ，那么它的子结点，从左至右，编号为  $4k+1$ ， $4k+2$ ， $4k+3$ ， $4k+4$ 。

采用四叉树编码的图像，可以用一个秘密的密码加密如下：每当进行细分时，其 4 个分支重新排序。对每个结点，重新排序可能有所不同，但根据密码和结点编号，其顺序是完全确定的。

不幸的是，有些人在编码程序中使用“保存密码”功能，并且在多个图像中使用相同的密码。通过观察精心挑选的测试图像的编码，发现使用相同密码编码的图像，不需要密码就可以解码。在这个测试图像中，每个像素的色饱和度不同，从 0 到  $n^2-1$ ，从左到右、自上而下按升序排列。当  $n=16$  时如图 4-6 所示。

你设法获取编码程序，并用它对测试图像编码。给出测试图像的四叉树编码，编写程序，对使用相同密码编码的图像进行解码。

**本题包含多组测试例！**

多组测试例的第一行是一个整数  $N$ ，然后是一个空行，接着是  $N$  个输入数据块。每个数据块的格式在问题描述中给出。每个数据块之间有一个空行。

输出格式包括  $N$  个输出数据块，每个输出数据块之间有一个空行。

**输入格式**

输入有多个测试例。输入的第一行是一个正整数，表示测试案例数。每个测试例的第一行是  $n$ ，接着是测试图像的四叉树编码，需要解码的秘密图像的四叉树编码。每个四叉树编码的第一行是一个正整数  $m$ ，表示该树的叶子结点数量。后面  $m$  行的格式如下：

$k$  intensity

表示结点编号  $k$  是一个叶子结点，相关联的色饱和度是 intensity。没有给出的结点是内部结点，或者不在四叉树里。可以假设色饱和度在 0~255 之间。还可以假设，根据以上描述的编码算法，输出的每个四叉树编码都是有效的。

```
1          *N
2          *测试例数
2          *n
4          *测试图像的四叉树编码
1 3
2 2
3 0
4 1
4          *秘密图像的四叉树编码
1 23
2 123
```

3 253

4 40

## 输出格式

对每个测试例，输出测试例编号，一个空行。然后，输出解码图像每个像素的色饱和度，每行对应图像中的一行。色饱和度的输出宽度为 4，右对齐，没有多余的空格。在测试例之间有一个空行。

## 【算法分析】

关于四叉树的相关知识，请参考网站：<http://baike.baidu.com/view/290215.htm>，北京大学“地理信息系统概论”课程：<http://www.jpku.pku.edu.cn/pkujpk/course/dlxxxt/ppt.htm>（第七章 空间数据管理）。代码参考自标程（<http://plg1.cs.uwaterloo.ca/~acm00/regionals/>）。

### （1）数据结构

存储四叉树的数组为

```
int tree[400][4];
```

存储叶子结点的数组为

```
int leaf[350];
```

图像的矩阵： $n \times n$  ( $1 \leq n \leq 16$ )。

叶子结点的数量为

```
int m;
```

### （2）根据叶子结点，构造四叉树结构

该功能由函数实现：

```
int processTree( int x )
```

形参是当前结点的编号。

在题目中详细介绍了构造四叉树的算法。本题中叶子结点的编号和相关联的值是已知的，构造时从根结点向叶子结点递归，当递归返回时产生内部结点的编号。

### （3）根据四叉树结构，对秘密图像解码

该功能由函数实现：

```
void buildImage( int x, int x1, int y1, int x2, int y2 )
```

其中形参  $x$  是当前结点，坐标对  $(x_1, y_1)$ ， $(x_2, y_2)$  是与结点  $x$  相关联的一片正方形区域，在该区域内，每个像素的色饱和度都是  $\text{leaf}[x]$ 。

然后根据四叉树结构，递归构造其他正方形区域。麻烦的地方是正方形区域的一对顶角坐标的计算。

### （4）输出解码后的图像信息

输出图像矩阵时，注意矩阵是转置的。

## 【程序代码】

---

程序名称：	zju1156.c
题    目：	Unscrambling Images
提交语言：	C
运行时间：	20ms
运行内存：	168KB

---



```

#include <stdio.h>
#include <memory.h>

int tree[400][4];           //存储四叉树
int leaf[350];             //存储叶子结点

//根据叶子结点, 构造四叉树结构
int processTree( int x ) {
    int v[4];               //4 个子结点
    int i, j, z;
    if( leaf[x] > -1 ) return leaf[x]; //x 是叶子结点
    //根据四叉树的定义, 递归构造四叉树
    for(i = 0; i < 4; i++)
        v[i] = processTree( 4*x+1+i );
    //构造结点 x 的 4 个子结点
    for(i = 0; i < 4; i++) {
        z = 0;
        for(j = 0; j < 4; j++)
            if( v[i]>v[j] ) z++;
        tree[x][z] = i;
    }
    return v[tree[x][0]];
}

int image[16][16];         //图像矩阵
//根据四叉树结构, 对秘密图像解码
void buildImage( int x, int x1, int y1, int x2, int y2 )
{
    int i, j;
    //当前结点是叶子结点
    if( leaf[x] > -1 )
    {
        //构造相关联的一片正方形区域, 其值都是 leaf[x]
        for( i = x1; i <= x2; i++ )
            for( j = y1; j <= y2; j++ )
                image[i][j] = leaf[x];
        return;
    }
    //根据四叉树结构, 递归构造其他正方形区域
    buildImage(4*x+1+tree[x][0], x1, y1, x1+(x2-x1)/2, y1+(y2-y1)/2 );
    buildImage(4*x+1+tree[x][1], x1+(x2-x1)/2+1, y1, x2, y1+(y2-y1)/2 );
    buildImage(4*x+1+tree[x][2], x1, y1+(y2-y1)/2+1, x1+(x2-x1)/2, y2 );
    buildImage(4*x+1+tree[x][3], x1+(x2-x1)/2+1, y1+(y2-y1)/2+1, x2, y2 );
}

int main() {

```

```

int i, j, k;
int n;                                     //n×n的图像
int m;                                     //叶子结点的数量
int intensity;                             //色饱和度

int N;                                     //数据块的数量
scanf("%d", &N);
while (N--)
{
    memset (tree, 0, sizeof(tree));
    int num;                               //测试例编号
    int iCase;                             //测试例的数量
    scanf("%d", &iCase );
    for (num=1; num<=iCase; num++) {
        scanf("%d%d", &n, &m);
        //读取测试图像的四叉树编码
        memset (leaf, 0xff, sizeof(leaf));
        for( i = 0; i < m; i++ ) {
            scanf("%d %d", &k, &intensity );
            leaf[k] = intensity;
        }
        //根据叶子结点, 构造四叉树结构
        processTree(0);

        //读取需要解码的秘密图像的四叉树编码
        scanf("%d", &m);
        memset (leaf, 0xff, sizeof(leaf));
        for( i = 0; i < m; i++ ) {
            scanf("%d %d", &k, &intensity );
            leaf[k] = intensity;
        }
        //根据四叉树结构, 对秘密图像解码
        buildImage(0, 0, 0, n-1, n-1);

        if( num>1 ) printf("\n");
        printf("Case %d\n\n", num );
        //输出解码后的图像信息
        for( i = 0; i < n; i++ ) {
            for( j = 0; j < n; j++ )
                printf("%4d", image[j][i] );
            printf("\n");
        }
        if (N) printf("\n");
    }
    return 0;
}

```

## 第五章 搜索算法题

在本章的题目主要是搜索算法题。需要一些搜索算法，如深度优先搜索、广度优先搜索算法等，实现题目的要求。

### ZJU1084- Channel Allocation<sup>[1、2、3]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768K

---

When a radio station is broadcasting over a very large area, repeaters are used to retransmit the signal so that every receiver has a strong signal. However, the channels used by each repeater must be carefully chosen so that nearby repeaters do not interfere with one another. This condition is satisfied if adjacent repeaters use different channels.

Since the radio frequency spectrum is a precious resource, the number of channels required by a given network of repeaters should be minimised. You have to write a program that reads in a description of a repeater network and determines the minimum number of channels required.

#### INPUT

The input consists of a number of maps of repeater networks. Each map begins with a line containing the number of repeaters. This is between 1 and 26, and the repeaters are referred to by consecutive upper—case letters of the alphabet starting with A. For example, ten repeaters would have the names A,B,C,...,I and J. A network with zero repeaters indicates the end of input.

Following the number of repeaters is a list of adjacency relationships. Each line has the form:

A:BCDH

which indicates that the repeaters B, C, D and H are adjacent to the repeater A. The first line describes those adjacent to repeater A, the second those adjacent to B, and so on for all of the repeaters. If a repeater is not adjacent to any other, its line has the form

A:

The repeaters are listed in alphabetical order.

Note that the adjacency is a symmetric relationship; if A is adjacent to B, then B is necessarily adjacent to A. Also, since the repeaters lie in a plane, the graph formed by connecting adjacent repeaters does not have any line segments that cross.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1084>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1129>

[3] <http://acm.uva.es/archive/nuevportal/data/problem.php?p=2243>

# OUTPUT

For each map (except the final one with no repeaters), print a line containing the minimum number of channels needed so that no adjacent channels interfere. The sample output shows the format of this line. Take care that channels is in the singular form when only one channel is required.

# SAMPLE INPUT

2	4	4
A:	A:BC	A:BCD
B:	B:ACD	B:ACD
	C:ABD	C:ABD
	D:BC	D:ABC
		0

# SAMPLE OUTPUT

1 channel needed.  
3 channels needed.  
4 channels needed.

# Problem Source:

South Africa 2001

# 【题目大意】

当无线电台在一个非常大的区域上传播信号时，为了每个接收器都能得到较强信号，使用转发器转发信号。然而，需要仔细地选择每个转发器使用的频道，以使附近的转发器不彼此干扰。如果邻近的转发器使用不同的频道，条件就得到满足。

因为无线电波的频谱是宝贵的资源，转发器所需频道的数量应减到最少。编程任务：读取转发器网络的描述信息，并计算出所需频道的最小使用量。

# 输入格式

输入包含许多转发器网络图。每幅图的第一行是转发器数目（1~26）。转发器用连续的大写字母表示，从 A 开始。例如，10 个转发器的名称分别是 A，B，C，…，I 和 J。当转发器的个数是 0 时，表示输入结束。

转发器数目之后，是其邻近关系的列表。每行的格式为

A:BCDH

表示转发器 B、C、D 和 H 与转发器 A 邻近。第一行描述与转发器 A 邻近的，第二行描述与 B 邻近的，直到描述完所有的转发器。如果某个转发器不与其他转发器相邻，它的形式为

A:

转发器依字母顺序列出。

注意：相邻是对称的关系；如果 A 与 B 相邻，那么 B 与 A 也相邻。因为转发器位于水平面内，由相邻的转发器构成的网络图没有相交的线。

# 输出格式

对于每幅图（除了最后一个没有转发器），输出一行，是转发器不互相干扰所需的最少频道数。输出格式参考样例输出。注意：频道数为 1 的话，“channel”为单数。

## 【算法分析】

本题的转发器网络相当于一个无向图,而邻近的转发器使用不同的频道相当于无向图的着色问题。关于无向图的着色问题,尤其是著名的四色问题<sup>[1、2、3]</sup>,在有关离散数学中都有详细的描述。

### (1) 样例分析

样例 2 和 3 的转发器网络图,如图 5-1 所示。

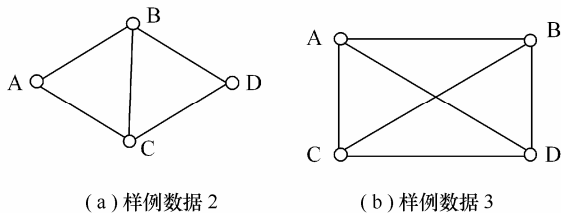


图 5-1 样例数据的转发器网络图

从图中看出,为确保邻近的转发器使用不同的频道,样例数据 2 需要使用 3 个频道(转发器 A、B 和 C 各不一样,转发器 D 的频道和 A 一样);样例数据 3 需要使用 4 个频道(因为这些转发器两两相邻)。

### (2) 构造无向图

使用数组  $g$  存储转发器网络的无向图:

```
bool g[26][26];
```

转发器的数量:

```
int n;
```

对每个转发器的邻接关系,对称地存储于数组  $g$  中即可。

### (3) 着色的实现

因为颜色不超过 4 种,所以有如下两种情况:

① 所有的转发器互不相邻,只要着一种颜色;

② 用两种或三种颜色去着色,看看是否能够成功着色。如果都不能实现着色,那就是 4 种颜色。

使用函数  $\text{dfs}()$  实现着色算法:

```
bool dfs(int id, int color)
```

形参  $i_d$  是起始着色的结点,  $\text{color}$  是限制的着色数量。

当某一结点  $i_d$  ( $0 \leq i_d < n$ ) 使用颜色  $i$  ( $0 \leq i < \text{color}$ ) 时,在已经着色的结点中,搜索是否使用过这种颜色。如果该颜色已经使用,则换一种颜色;否则搜索下一个结点。当所有的结点着色完成,返回成功标志  $\text{true}$ ;当颜色数  $\text{color}$  都用完了,还不能给所有结点着色,则返回失败标志  $\text{false}$ 。

[1] <http://tieba.baidu.com/f?kz=152386896>

[2] <http://blog.csdn.net/Feisy/archive/2008/04/25/2328692.aspx>

[3] <http://www.cs.sysu.edu.cn/~jhuang/DiscreteMath/Chapter15-18.ppt>

## 【程序代码】

---

程序名称:	zju1084.cpp
题    目:	Channel Allocation
提交语言:	C++
运行时间:	00:00.00
运行内存:	436K

---

```
#include <stdio.h>
#include <memory.h>

bool g[26][26];           //存储无向图
int used[26];             //已经使用的颜色
int n;                    //转发器的数量

//深度优先搜索，实现着色
//形参 id 是起始着色的结点，color 是限制的着色数量
bool dfs(int id, int color)
{
    int i, j;
    bool flag;             //着色成功的标志
    //在规定的颜色数中着色
    for (i = 0; i < color; i++)
    {
        flag = true;
        //结点 id 使用第 i 号颜色
        used[id] = i;
        //判断相邻结点是否使用了该颜色
        for (j = 0; j < id; j++)
            if (g[id][j] && used[id] == used[j])
            {
                //该颜色已经使用，换一种颜色
                flag = false;
                break;
            }
        //该颜色有效，当所有结点着色完毕返回 true，或者给下一个结点着色
        if (flag && (id == n-1 || dfs(id+1, color)))
            return true;
    }
    //使用 color 个颜色没有实现全部着色
    return false;
}

int main()
{
    int i, j;
    bool one;               //只需要一种颜色的标志
```

```

char adjacent[50];
while (scanf("%d", &n) && n)
{
    memset(g, 0, sizeof(g));
    memset(used, 0, sizeof(used));
    //构造无向图
    one = true;
    for ( i = 0; i < n; i++) {
        scanf("%s", adjacent);
        //从第 3 个字符起, 是邻接关系结点
        for (j = 2; adjacent[j]; j++, one = false)
        {    //对称
            g[i][adjacent[j] - 'A'] = true;
            g[adjacent[j] - 'A'][i] = true;
        }
    }
    if (one)                                //1 种颜色
        printf("1 channel needed.\n");
    else if (dfs(1, 2))                      //2 种颜色
        printf("2 channels needed.\n");
    else if (dfs(1, 3))                      //3 种颜色
        printf("3 channels needed.\n");
    else                                     //4 种颜色
        printf("4 channels needed.\n");
}
return 0;
}

```

## ZJU1085-Alien Security<sup>[1、2、3]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768K

---

You are in charge of security at a top-secret government research facility. Recently your government has captured a live extra-terrestrial (ET) life form, and is hosting an open day for fellow researchers. Of course, not all the guests can be trusted, so they are assigned different security clearance levels. Only guests with a level 5 rating will be allowed into the lab where the extra-terrestrial is being held; other than that, everyone is free to roam throughout the rest of the facility. Each room in the facility is connected via one-way airlocks, so that you can pass through the

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1085>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1130>

[3] <http://acm.uva.es/archive/nuevportal/data/problem.php?p=2244>

door in only one direction.

To protect your precious ET you will put in place enhanced security measures (in the form of armed guards) on the route leading to the room containing the ET, but not in the room itself-the guards do not have sufficient clearance to enter the room containing the ET.

The guards will check the identity and the security rating of all guests trying to pass through the room in which they are stationed, so you would like to place the guards where they will cause the minimum amount of irritation to the guests who have no intention of visiting the ET. The room where the guards must be placed thus satisfies the following two conditions:

1. In order to get to the room containing the ET, the guests must pass through the room containing the guards;
2. There is no other room with this property that is closer to the room containing the ET-remember, the guards cannot be placed in the room containing the ET itself.

The diagram below illustrates one possible map of your facility:

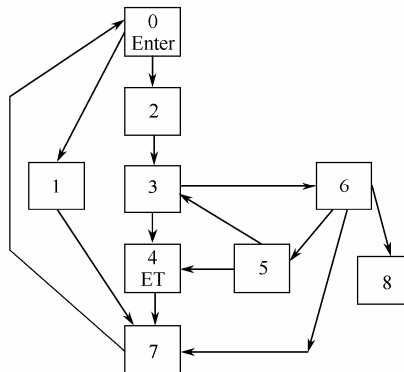


Figure 5-2 The diagram

Note that placing the guards in room 2 would satisfy the first condition, but room 3 is closer to the ET, so the guards must be placed in room 3.

All guests enter through room 0, the entrance to your facility. Your program accepts a sequence of lines containing integers. The first line consists of two integers: the number of rooms, and the room in which the ET is being held (out of his own free will, of course).

The rest of the input is a sequence of lines consisting of only two integers, specifying where the airlock-doors are located. The first number on these lines specifies the source room, and the second the destination room. Remember: you can pass only from the source room to the destination room.

The output of your program consists only of a single line:

Put guards in room  $N$ .

where  $N$  is the room you've decided to place the guards.

## This problem contains multiple test cases!

The first line of a multiple input is an integer  $N$ , then a blank line followed by  $N$  input blocks. Each input block is in the format indicated in the problem description. There is a blank line between



input blocks.

The output format consists of  $N$  output blocks. There is a blank line between output blocks.

## SAMPLE INPUT

This input sequence specifies the map shown above.

```
1
9 4
0 2
2 3
3 4
5 3
5 4
3 6
6 5
6 7
6 8
4 7
0 1
1 7
7 0
```

## SAMPLE OUTPUT

```
Put guards in room 3.
```

## Problem Source

South Africa 2001

### 【题目大意】

你在一家高度机密的政府研究机构负责安全事务。最近，你所在的政府抓获了一个外星人(ET)，正在为同行研究者举办一个开放日。当然，不是所有的客人都被信赖的，因此，要给客人分配不同的安全许可证级别。只有评定为 5 级的客人才能进入放有外星人的实验室；除此之外，每个人可以在研究机构的其他地方随意参观。研究机构的每个房间由单向密封过渡舱连接，所以你进去后只能向一个方向通过。

为了保护珍贵的外星人(ET)，在通往存放外星人房间的路上，将加强安全措施（以武装守卫的形式），但并不包括那房间本身——守卫没有足够的安全级别进入该房间。

对于要通过守卫室的所有客人，守卫要检查他们的 ID 和安全级别。为了减少不能观看外星人客人的刺激，你得安排好守卫的房间。设置守卫的房间必须满足以下两个条件：

- (1) 要参观到外星人，客人必须通过守卫的房间；
- (2) 没有其他房间更接近放有外星人的房间——记住，守卫不能设在有外星人的房间内。

下面是其中一份研究机构的地图：

注意：把守卫设在房间 2 满足第一个条件，但房间 3 更接近 ET 房间，因此，守卫应设在房间 3 内。

所有客人首先进入房间 0，这是通往研究机构的入口。程序的输入有多行整数，第一行有

两个整数：房间的数目，ET 所在的房间（自然不是 ET 自定的）。

其余各行也只有两个整数，指明密封过渡舱的门通向哪里。第一个数是源房间，第二个数是目标房间。记住：只能由源房间通向目标房间。

程序的输出只有一行：

Put guards in room *N*.

*N* 是设置守卫的房间。

本题包含多组测试数据！

输入的第一行是一个整数 *N*，然后是空行，接着是 *N* 组测试数据。每组测试数据的格式与问题描述中的一样。组数据之间有一个空行。

输出有 *N* 组数据，各组之间有一个空行。

### 样例输入

样例输入的数据，就是题目中所表示的图。

### 【算法分析】

本题是搜索题，不仅用到了广度优先搜索，而且用到了深度优先搜索。

#### （1）构造邻接矩阵

使用数组 **data** 表示研究机构的邻接矩阵：

```
const int MaxN = 100;
bool data[MaxN][MaxN];
```

房间的数目是变量 **n**，ET 所在的房间是变量 **et**。

在读取数据上，要处理好空行的问题。为此只能将房间之间的关系当作字符串输入，当读到一个字符串，其内容为空时，表示读到了空行。从读取的字符串中输入数据，可以使用 `sscanf()` 函数，其使用方法与 `scanf()` 基本上是一样的。

#### （2）广度优先搜索，搜索各个房间到 ET 房间的最短距离

虽然可以使用 Floyd 方法，计算各个房间之间的最短距离，因为很多计算结果都没有用，效率就太低了。如果站在 ET 的房间看，将密封过渡舱的方向反一下，就成了单源最短路径（Shortest Path Faster Algorithm, SPFA）问题，这时使用广度优先搜索的算法，可以得到很高的效率。实现的函数为

```
void shortest()
```

实现广度优先搜索算法的重要一步是建立队列，可以使用链表、数组、C++ 的 `string` 和 `vector()` 模拟队列，最简便的方法是使用 C++ 的标准模板库 `queue()`：

```
queue<int> q;
```

从当前房间 *x* 向四周扩展，如果到房间 *i* 有通路（是原始数据的反向，即 `data[i][x]`），且能够获得更短的距离，则经过房间 *i*，并将房间 *i* 入栈。

对于样例数据，其他房间到房间 ET 的最短路径（数组 **dis**）如表 5-1 所示。

表 5-1 样例数据的最短路径

下标	0	1	2	3	4	5	6	7	8
距离	3	5	2	1	0	1	2	4	∞

#### （3）枚举，除了房间 ET 外，逐个房间设置守卫室，搜索最优解

要满足设置守卫房间的两个条件，简单的办法是枚举。既然守卫的房间是必经之路，假设

房间  $i$  ( $0 \leq i \leq n-1$ ,  $i \neq \text{et}$ ) 是守卫的房间, 那么去掉房间  $i$ , 从入口 0 就不能到达 ET 房间。这样的房间可能有很多, 再根据前面计算的最短路径, 取最短路径最小的那个房间。

去掉房间  $i$ , 从入口 0 能否能到达 ET 房间, 实现的函数是:

```
int search(int id)
```

形参  $\text{id}$  是当前正在搜索的房间编号。

这只需追踪从房间 ID 到房间 ET 之间是否有路。如果  $\text{id} = \text{et}$ , 说明有路到达房间 ET。

## 【程序代码】

---

程序名称:	zju1085.cpp
题    目:	Alien Security
提交语言:	C++
运行时间:	00:00.00
运行内存:	852K

---

```
#include <iostream>
#include <queue>
using namespace std;

const int MaxN = 100;
int n, et;                //房间的数目, ET 所在的房间
bool used[MaxN];          //DFS 搜索标志, 已经搜索过的房间
bool data[MaxN][MaxN];    //表示研究机构的邻接矩阵
int dis[MaxN];            //其他房间到 ET 房间的距离
```

//广度优先搜索, 搜索各个房间到 ET 房间的最短距离

```
void shortest() {
    queue<int> q;          //搜索队列
    q.push(et);           //起始搜索位置
    dis[et] = 0;
    int x;
    //队列不为空
    while (!q.empty()) {
        x = q.front();     //取队列头结点
        q.pop();
        for (int i = 0; i < n; ++i)
            //注意, 从 ET 出发是逆向行走
            //经过房间 x 到 ET 比从房间 i 到 ET 更近
            if (data[i][x] && dis[x] + 1 < dis[i]) {
                q.push(i);
                dis[i] = dis[x] + 1;
            }
    }
}
```

//深度优先搜索, 从房间 0 出发, 能否到达 ET 房间

```

int search(int id){
    if (id == et) return 1;           //成功到达 ET 房间
    used[id] = 1;                     //房间 id 已经搜索
    //搜索下一个出口
    for (int i = 0; i < n; ++i)
        //房间 i 还没有搜索, 且房间 id 与房间 i 之间有边
        if (!used[i] && data[id][i])
            if (search(i)) return 1;
    return 0;
}

int main(){
    int cases;                        //测试例数
    int room;                          //守卫的房间
    scanf("%d", &cases);
    for (int t = 0; t < cases; ++t){
        scanf("%d%d\n", &n, &et);
        memset(data, 0, sizeof(data));
        for (int i = 0; i < MaxN; ++i)
            dis[i] = INT_MAX;
        //构造研究机构的邻接矩阵
        char line[10];
        int a, b;
        while (gets(line)){
            if (strcmp(line, "")==0) break;
            sscanf(line, "%d%d", &a, &b);
            data[a][b] = true;         //房间 a, b 之间有边
        }
        //广度优先搜索, 搜索各个房间到 ET 房间的距离
        shortest();
        //枚举, 除了房间 ET 外, 逐个房间设置守卫室, 搜索最优解
        //房间 room 到房间 ET 之间的距离为 d
        int d = dis[0];
        room = 0;
        for (int i = 1; i < n; ++i){
            if (i == et) continue;
            memset(used, 0, sizeof(used));
            //设置在房间 i, 也就是在图中将房间 i 拿掉
            used[i] = 1;
            //从入口 0 不能到达 ET, 说明房间 i 是必经之路
            //而且获得到房间 ET 的更短距离
            if (!search(0) && dis[i] < d){
                room = i;
                d = dis[i];
            }
        }
        if (t) cout << endl;         //组之间的空行
    }
}

```

```

        cout << "Put guards in room " << room << "." << endl;
    }
    return 0;
}

```

## ZJU1091-Knight Moves<sup>[1, 2, 3]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768K

---

A friend of you is doing research on the Traveling Knight Problem (TKP) where you are to find the shortest closed tour of knight moves that visits each square of a given set of  $n$  squares on a chessboard exactly once. He thinks that the most difficult part of the problem is determining the smallest number of knight moves between two given squares and that, once you have accomplished this, finding the tour would be easy.

Of course you know that it is vice versa. So you offer him to write a program that solves the "difficult" part.

Your job is to write a program that takes two squares  $a$  and  $b$  as input and then determines the number of knight moves on a shortest route from  $a$  to  $b$ .

### Input Specification

The input file will contain one or more test cases. Each test case consists of one line containing two squares separated by one space. A square is a string consisting of a letter (a~h) representing the column and a digit (1~8) representing the row on the chessboard.

### Output Specification

For each test case, print one line saying "To get from  $xx$  to  $yy$  takes  $n$  knight moves."

### Sample Input

```

e2 e4
a1 b2
b2 c3
a1 h8
a1 h7
h8 a1
b1 c3
f6 f6

```

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1091>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2243>

[3] <http://acm.uva.es/p/v4/439.html>

## Sample Output

```
To get from e2 to e4 takes 2 knight moves.  
To get from a1 to b2 takes 4 knight moves.  
To get from b2 to c3 takes 2 knight moves.  
To get from a1 to h8 takes 6 knight moves.  
To get from a1 to h7 takes 5 knight moves.  
To get from h8 to a1 takes 6 knight moves.  
To get from b1 to c3 takes 1 knight moves.  
To get from f6 to f6 takes 0 knight moves.
```

## Problem Source:

University of Ulm Local Contest 1996

### 【题目大意】

你的一位朋友正在研究骑士旅行问题（TKP）。在一个有  $n$  个方格的棋盘上，你得找到一条最短的封闭骑士旅行的路径，使能够遍历每个方格一次。他认为问题的最困难部分在于，对两个给定的方格，确定骑士移动所需的最小步数。你曾经解决过这类问题，找到这个路径应该不困难。

当然，你知道反之亦然，所以你帮助他编写一个程序，解决这个“困难的”部分。

**编程任务：**输入有两个方格  $a$  和  $b$ ，确定骑士在最短路径上从  $a$  到  $b$  移动的次数。

### 输入格式

输入包含一组或多组测试例。每个测试例一行，是两个方格，用空格隔开。棋盘上的一个方格用一个字符串表示，字母（ $a\sim h$ ）表示列，数字（ $1\sim 8$ ）表示行。

### 输出格式

对每个测试例，输出一行："To get from  $xx$  to  $yy$  takes  $n$  knight moves."

### 【算法分析】

这是一道经典的题目，可以在网上看到多种实现算法。主要有深度优先搜索、广度优先搜索和 Floyd 算法。通过本题的练习，相信在最短路径的算法方面会有很大的收获。

最容易的算法是深度优先搜索，从起点向 8 个方向递归，计算从该起点到棋盘上所有位置的最短路径。这个算法最致命的缺点是速度慢，因为计算的工作量太大。从 ZOJ 在线测试的运行时间看，差点超时。如果数据稍微多一点，或者棋盘稍微大一点，超时就不可避免。

接着容易让人想到的算法就是广度优先搜索算法。该算法也是从起点向 8 个方向搜索，但不是递归的方式，而是建立一个队列，将待搜索的结点放到队列里面，已经搜索的结点从队列里面删除，由于没有重复搜索，效率自然就高很多了。从 ZOJ 在线测试的运行时间看，比深度优先搜索算法要快多了。

本题由于棋盘很小，所以就有人想到，为什么不把所有格子之间的最短路径所需的步数全部计算出来呢？这样对所有的输入，然后查表输出就可以了，速度自然很快。这要分两步进行：

（1）骑士走一步可以到哪些位置

由于棋盘是  $8\times 8$  的，在任意一个方格，骑士都可以走到棋盘上的任意其他方格，这样就有 64 种可能性；因为有 64 格，所以要保存任意两格之间的最短路径的步数，需要  $64\times 64$  的

数组：

```
int knight[64][64]
```

现在就需要建立  $64 \times 64$  的矩阵数组与棋盘之间的对应关系。设矩阵数组的单元  $(i, j)$  ( $0 \leq i, j \leq 63$ )，对应棋盘上的骑士从方格坐标为  $a(i/8, i\%8)$  跳到方格坐标  $b(j/8, j\%8)$ ，如果只需要走一步时为 1，否则为  $\infty$ （取 10 就可以了）；方格自身为 0。

设  $(x, y)$  为方格  $a$  和  $b$  之间的坐标差值，其计算公式是：

```
x = i/8-j/8;
```

```
y = i%8-j%8;
```

在棋盘上  $(x, y)$  的相对位置有 8 个方向，如图 5-3 所示的阴影单元格：

	(-1, 2)		(1, 2)	
(-2, 1)				(2, 1)
		(0, 0)		
(-2, -1)				(2, -1)
	(-1, -2)		(1, -2)	

图 5-3 棋盘的行走方向

为了简化代码的编写，采用 `abs()` 函数表示：

```
x = abs(i/8-j/8);
```

```
y = abs(i%8-j%8);
```

明显看到，当  $x$  方向变化为 1 时， $y$  方向变化为 2；或者  $x$  方向变化为 2 时， $y$  方向变化为 1。因此就有：

```
if (x==1 && y==2 || x==2 && y==1)
```

```
k[i][j] = k[j][i] = 1;
```

数组  $k$  的结果，参见文件 `zju1091-knightMoves.xls`。

(2) 计算其他的单元格，即棋盘格子之间的最短路径所需的步数大于 1

由于要计算所有单元格之间的最短路径，当然是采用 Floyd 算法。Floyd 算法的具体描述请见 ZJU1053-FDNY to the Rescue！在一般的数据结构和计算方法的书籍中都有。

从 ZOJ 在线测试的运行时间看，这个方法最快。该方法适合计算棋盘较小、输入数据量大的情况，而广度优先搜索适合计算棋盘较大、输入数据量小的情况。

## 【程序代码】

(1) 深度优先搜索算法：

程序名称： zju1091-dfs.c

题 目： Knight Moves

提交语言： C

运行时间： 00:00.87

运行内存： 392K

```
#include <stdio.h>
```

```
#include <memory.h>
```

```
int knight[8][8];
```

```
//棋盘
```

```

int x[] = {1,1,2,2,-1,-1,-2,-2};           //8 个方向的坐标增量
int y[] = {2,-2,1,-1,2,-2,1,-1};

//深度优先搜索算法，搜索最短路径
void DFS(int si, int sj, int moves)
{
    //边界控制
    if(si<0 || sj<0 || si>=8 || sj>=8 || moves>=knight[si][sj]) return;
    knight[si][sj] = moves;                  //最优解
    //分别向 8 个方向递归搜索
    int i;
    for (i=0; i<8; i++)
        DFS(si+x[i], sj+y[i], moves+1);
}

int main()
{
    char a[10], b[10];
    while(scanf("%s%s", a, b)!=EOF)
    {
        memset(knight, 10, sizeof(knight));
        DFS(a[0]-'a', a[1]-'1', 0);
        printf("To get from %s to %s takes %d knight moves.\n",
               a, b, knight[b[0]-'a'][b[1]-'1']);
    }
    return 0;
}

```

## (2) 广度优先搜索算法:

---

程序名称:	zju1091-bfs.cpp
题 目:	Knight Moves
提交语言:	C++
运行时间:	00:00.16
运行内存:	1016K

---

```

#include <iostream>
#include <queue>
using namespace std;

//结点的数据结构
struct point{
    int x,y;           //棋盘坐标
    int c;             //最短路径
}from, to;

int main(){
    queue<point> q;     //优先队列

```



```

char src[3], dist[3];
//8 个方向的坐标增量
int dx[] = {1,1,2,2,-1,-1,-2,-2};
int dy[] = {2,-2,1,-1,2,-2,1,-1};
while(scanf("%s%s", src, dist)!=EOF){
    printf("To get from %s to %s ", src, dist);
    while(!q.empty()) q.pop();          //队列清空
    //起始点
    from.x = src[0]-'a';
    from.y = src[1]-'1';
    from.c = 0;
    //目标点
    to.x = dist[0]-'a';
    to.y = dist[1]-'1';
    //第 1 个结点进入队列
    q.push(from);
    point temp;
    while(true){
        from = q.front();                //取出首结点
        q.pop();
        //到达目标结点
        if (from.x==to.x && from.y==to.y) break;
        //向 8 个方向扩展搜索
        for(int i=0; i<8; i++){
            temp.x = from.x + dx[i];
            temp.y = from.y + dy[i];
            temp.c = from.c + 1;
            //边界控制
            if(temp.x<0 || temp.x>7 || temp.y<0 || temp.y>7) continue;
            //满足条件的结点入栈
            q.push(temp);
        }
    }
    printf("takes %d knight moves.\n", from.c);
}
}

```

### (3) Floyd 算法:

---

程序名称:	zju1091.c
题 目:	Knight Moves
提交语言:	C
运行时间:	00:00.01
运行内存:	412K

---

```

#include <stdio.h>
int i, j, m;

```

```

//计算棋盘上任意两格之间的最短路径
void shortestest(int k[][64])
{
    //计算骑士走一步可以到达的位置
    int x, y;
    for(i = 0; i < 64; k[i][i] = 0, ++i)
        for(j = 0; j < 64; ++j) {
            //棋盘上两格之间的相对位置
            x = abs(i/8-j/8);
            y = abs(i%8-j%8);
            if (x==1 && y==2 || x==2 && y==1)
                k[i][j] = k[j][i] = 1;           //只需要跳一步
        }
    //通过 Floyd 算法, 计算棋盘上其他两格之间的最短路径
    for(m = 0; m < 64; ++m)
        for(i = 0; i < 64; ++i)
            for(j = 0; j < 64; ++j)
                if(k[i][m] + k[m][j] < k[i][j])
                    k[i][j] = k[i][m] + k[m][j];
}

int main()
{
    //保存棋盘任意两点之间的最短路径
    int knight[64][64];
    //初始化为∞
    for(i = 0; i < 64; ++i)
        for(j = 0; j < 64; ++j)
            knight[i][j] = 10;
    shortestest(knight);

    char s[5], t[5];
    while (scanf("%s%s", s, t)!=EOF)
    {
        int x = (s[0]-'a')*8+(s[1]-'1');
        int y = (t[0]-'a')*8+(t[1]-'1');
        printf("To get from %s to %s ", s, t);
        //直接查表输出结果
        printf("takes %d knight moves.\n", knight[x][y]);
    }
    return 0;
}

```

# ZJU1101-Gamblers<sup>[1]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768 KB

---

A group of  $n$  gamblers decide to play a game:

At the beginning of the game each of them will cover up his wager on the table and the assistant must make sure that there are no two gamblers have put the same amount. If one has no money left, one may borrow some chips and his wager amount is considered to be negative. Assume that they all bet integer amount of money.

Then when they unveil their wagers, the winner is the one whose bet is exactly the same as the sum of that of 3 other gamblers. If there are more than one winner, the one with the largest bet wins.

For example, suppose Tom, Bill, John, Roger and Bush bet \$2, \$3, \$5, \$7 and \$12, respectively. Then the winner is Bush with \$12 since  $2+3+7=12$  and it's the largest bet.

## Input

Wagers of several groups of gamblers, each consisting of a line containing an integer  $1 \leq n \leq 1000$  indicating the number of gamblers in a group, followed by their amount of wagers, one per line. Each wager is a distinct integer between  $-536870912$  and  $+536870911$  inclusive. The last line of input contains 0.

## Output

For each group, a single line contains the wager amount of the winner, or a single line containing "no solution".

## Sample Input

5	5	0
2	2	
3	16	
5	64	
7	256	
12	1024	

## Output for Sample Input

```
12
no solution
```

### 【 题目大意 】

$n$  个赌徒决定这样玩一场游戏：

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1101>

游戏开始时：他们各自将自己桌上的赌注盖住，同时助手必须保证任何两个赌徒的赌注是不相同的。如果其中一个赌徒没有钱了，那他可以借些筹码，当然他的赌注就是负数了。假设：他们的筹码都是整数。

然后，他们揭开所有的赌注。赢家：他的赌注是其他三个人赌注的总和。如果赢家不止一个，那么拥有最大赌资的人是赢家。

举例来说，假如 Tom, Bill, John, Roger and Bush 分别打赌\$2、\$3、\$5、\$7 和\$12。因为  $\$2 + \$3 + \$7 = \$12$  且其是最大赌注，所以赢家是 Bush。

### 输入格式

有多组赌徒。对每组赌徒，第一行是  $n(1 \leq n \leq 1000)$ ，表示赌徒的人数，接下来每行是一个赌徒的赌资。赌资是互不相同的整数：-536870912~+536870911。输入的最后一行是 0。

### 输出格式

对于每组赌徒，输出一行，是赢家的赌资或 “no solution”。

### 【算法分析】

这是一个枚举类型的题目，在所有的数字中找出三个数字  $n_1, n_2, n_3$ ，其和是另一个数字  $n$ 。如果有多组这样的数字，则要确保答案  $n$  是最大的数字。

将所有的数字按从小到大的顺序排序；然后从序列的后面取数作为数字  $n$ ，则自然能够保证数字  $n$  是最大的数字。有了数字  $n$ ，另外的三个数字  $n_1, n_2, n_3$ ，就得靠枚举了。

#### (1) 数据结构

```
int n;                //每组赌徒的数量，也就是数字的个数；
int bet[1000];        //每个赌徒的赌注；
int winner;           //赢家的编号，其赌注是 bet[winner]
```

#### (2) 将所有的数字按从小到大的顺序排序

利用 C 语言库函数：

```
qsort(bet, n, sizeof(int), comp);
```

其中 **comp** 为比较因子，按升序排序：

```
int comp(const void *a, const void *b)
{
    return (*(int *)a - *(int *)b);
}
```

#### (3) 枚举，找出赢家的编号

在排序的序列中，从后往前取赢家的序号  $i$ ，这样计算的结果自然是赌注最大的赢家。

三个赌徒的编号，至少有一个编号在赢家的前面 ( $j$ )。由于存在负数，另外两个赌徒的编号，就有可能在赢家编号的后面。先枚举两个赌徒 ( $j, k$ )，则第三个赌徒的赌注 **minus** 为

```
minus = bet[i] - bet[j] - bet[k];
```

可以在序列中遍历查找 **minus**，由于是有序序列，采用二分搜索就可以了。如果数字 **minus** 存在，则序号  $i$  就是赢家；否则继续查找。直到所有的数字都查找完毕，都没有赢家的话，就是没有解答了。

#### (4) 二分搜索算法的实现

二分搜索算法是通过函数 **bSearch** 实现的：

```
int bSearch(int k, int val);
```

程序中形参  $k$  是搜索的起始位置， $val$  是要搜索的数值。

关于二分搜索算法的实现，请参看有关数据结构的书籍。

(5) 为什么要采用排序和二分搜索的办法？

排序的目的有两个：①能够实现二分搜索；②如果存在多个赢家，由于从后往前取赢家编号，则只需要查找一个赢家，而且能够确保是最大的赢家。采用二分搜索方法，比一次遍历查找要快多了。

如果采用强力枚举，找到一个解（还不一定是赌注最大的）需要 4 重循环，算法时间复杂度是  $O(n^4)$ 。而将赌注排序之后，并采用二分搜索的方法，虽然是 3 重循环加二分搜索的时间，算法时间复杂度是  $O(n^3 \lg n)$ ，却能够确保是最大的赢家。

### 【程序代码】

---

程序名称： zju1101.c

题 目： Gamblers

提交语言： C

运行时间： 30ms

运行内存： 168KB

---

```
#include <stdio.h>
#include <stdlib.h>

int n; //每组赌徒的数量，也就是数字的个数
int bet[1000]; //每个赌徒的赌注
int winner; //赢家的编号

//排序函数的比较因子，升序排序
int comp(const void *a, const void *b)
{
    return (*(int *)a - *(int *)b);
}

//二分搜索算法，从序列 bet[k]~bet[n]中查找数字 val
int bSearch(int k, int val)
{
    int left, right, mid;
    left = k+1; //计算左边界
    right = n-1; //计算右边界
    while(left<=right)
    {
        mid = (left + right)/2; //中间位置
        if(bet[mid] == val) return mid; //找到了，返回位置序号
        else if(bet[mid]<val) left = mid+1; //继续查找
        else right = mid-1;
    }
    return 0; //没有找到
}
```

```

//枚举算法，求解答案
int solve()
{
    int i, j, k;
    int minus, pos;
    qsort(bet, n, sizeof(int), comp);    //将所有的数字按从小到大的顺序排序
    //枚举查找
    for(i=n-1; i>0; i--)                //赢家从序列的后面取，确保是最大的
        for(j=0; j<i; j++)              //枚举两个数
            for(k=j+1; k<n; k++)
            {
                minus = bet[i]-bet[j]-bet[k]; //计算第三个数
                pos = bSearch(k, minus);      //二分查找第三个数
                if(pos && pos!=i)              //序列中存在第三个数
                {
                    winner = i;               //赢家找到了
                    return 1;
                }
            }
    return 0;                            //没有赢家
}

int main()
{
    int i;
    while(scanf("%d", &n) && n)
    {
        for(i=0; i<n; i++)
            scanf("%d", &bet[i]);
        if(solve()) printf("%d\n", bet[winner]); //找到了，输出赢家的赌注
        else printf("no solution\n");           //没有找到
    }
    return 0;
}

```

## ZJU1103-Hike on a Graph<sup>[1]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

"Hike on a Graph" is a game that is played on a board on which an undirected graph is drawn. The graph is complete and has all loops, i.e. for any two locations there is exactly one arrow between

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1103>

them. The arrows are coloured. There are three players, and each of them has a piece. At the beginning of the game, the three pieces are in fixed locations on the graph. In turn, the players may do a move. A move consists of moving one's own piece along an arrow to a new location on the board. The following constraint is imposed on this: the piece may only be moved along arrows of the same colour as the arrow between the two opponents' pieces.

In the sixties ("make love not war") a one-person variant of the game emerged. In this variant one person moves all the three pieces, not necessarily one after the other, but of course only one at a time. Goal of this game is to get all pieces onto the same location, using as few moves as possible. Find out the smallest number of moves that is necessary to get all three pieces onto the same location, for a given board layout and starting positions.

### Input Specification

The input file contains several test cases. Each test case starts with the number  $n$ . Input is terminated by  $n=0$ . Otherwise,  $1 \leq n \leq 50$ . Then follow three integers  $p_1, p_2, p_3$  with  $1 \leq p_i \leq n$  denoting the starting locations of the game pieces. The colours of the arrows are given next as a  $n \times n$  matrix of whitespace-separated lower-case letters. The element  $m_{ij}$  denotes the colour of the arrow between the locations  $i$  and  $j$ . Since the graph is undirected, you can assume the matrix to be symmetrical.

### Output Specification

For each test case output on a single line the minimum number of moves required to get all three pieces onto the same location, or the word "impossible" if that is not possible for the given board and starting locations.

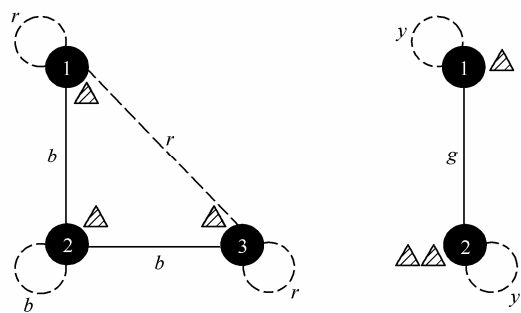


Figure 5-4

#### Sample Input

```
3 1 2 3
r b r
b b b
r b r
2 1 2 2
y g
g y
0
```

#### Sample Output

```
2
impossible
```

# Problem Source

University of Ulm Local Contest 2000

## 【题目大意】

“Hike on a Graph”是一种在木板上玩的游戏，木板上有一幅无向图。图是完整的并画有所有回路，即任意两个位置之间都有一条标线，标线有各种不同的颜色。有三个游戏者，他们各有一个棋子。游戏开始时，这三个棋子放在图上的固定位置。游戏者需要按照顺序移动棋子。每次移动只要沿着板上无向图的标线从现在的位置挪到新位置。限制如下：当标线的颜色与两个对手棋子之间的标线颜色一样时，棋子才能移动。

在 20 世纪 60 年代，有一款单人玩的游戏（make love not war）。在这款游戏里，一个人要移动三个棋子，不必一个接着一个移动，但每次只能移动一个棋子。游戏的任务是：用最少的步数，将所有的棋子移动到同一个位置上。根据木板上图的布局及棋子的开始位置，计算出最少的步数，把所有的棋子都移动到同一个位置上。

## 输入格式

输入有多组数据。对每组数据，第一个数据是  $n$  ( $1 \leq n \leq 50$ )。当  $n=0$  时输入结束。然后是三个整数  $p_1, p_2, p_3$  ( $1 \leq p_i \leq n$ )，表示棋子的初始位置。标线颜色由矩阵  $n \times n$  表示，都是小写字母（字母之间用空格隔开）。矩阵元素  $m_{ij}$  表示位置  $i$  和  $j$  之间的标线颜色。因为游戏图是无向的，你可以认为矩阵是对称的。

## 输出格式

对每组测试数据，输出一行：是把所有的棋子都移动到同一个位置上，所需要的最少的步数。如果无法实现目标，只需输出 “impossible”。

## 【算法分析】

本题的解答选自标程<sup>[1]</sup>，在该网页上还有裁判的算法分析，见光盘文件：Hike on a Graph.doc。

给定一个无向完全图，每条边都有一个颜色。游戏开始时，有三个棋子在图的三个结点上。每次可以沿着一条边移动任意一个棋子，要求该边的颜色和另外两个棋子之间边的颜色相同。要求用最少的步数，将所有的棋子移动到同一个位置上。

采用广度优先搜索算法，能够很快判定目标能否实现，以及计算实现目标所需要的最少步骤。

### （1）样例分析

对第一组数据，各个结点之间边的颜色如图 5-5 所示。

结点	1	2	3
1	r	b	r
2	b	b	b
3	r	b	r

图 5-5 无向图中边颜色的邻接矩阵

[1] <http://www.informatik.uni-ulm.de/acm/Locals/2000/html/judge.html>



从图中看出，结点本身存在自循环，是有边的。如结点 1，自循环边的颜色是 r。

结点 2、3 上有棋子，结点之间的颜色是 b，可以把结点 1 上的棋子移到结点 2（边的颜色是 b）；结点 2 自循环的颜色是 b，可以把结点 3 上的棋子移到结点 2（边的颜色是 b）。这样 2 步就把棋子都移到结点 2 上面了。

对第二组数据，结点自循环的颜色与相邻结点之间边的颜色不一样，棋子无法移动，所以是 “impossible”。

## （2）数据结构

无向图中边的颜色用数组表示：

```
char g[51][51];
```

无向图中结点的数量：

```
int n;
```

3 个棋子的初始位置：

```
int a,b,c;
```

能否实现目标：

```
int flag;
```

flag=1 表示能够实现目标，否则不能实现目标。

## （3）广度优先搜索算法的实现

三个棋子到达不同位置时所需要的步骤，用数组 step 保存：

```
unsigned int step[51][51][51];
```

使用数组 list 存储广度优先搜索算法所需要的链表：

```
struct str{
    char a,b,c;
}list[51*51*51];
```

链表头尾结点的编号：

```
int h = 0;
```

```
int t = 0;
```

关于广度优先搜索算法的原理，请参考有关数据结构或者算法分析的书籍。

在算法中，三个结点分别移动，直到链表搜索完毕或者三个结点是同一个结点为止。具体的实现算法，请看下面的代码注释。

## 【程序代码】

---

程序名称： zju1103.c

题 目： Hike on a Graph

提交语言： C

运行时间： 10ms

运行内存： 160KB

---

```
#include <stdio.h>
#include <memory.h>
#include <ctype.h>
```

```
int main()
{
```

```

char g[51][51]; //无向图中边的颜色
unsigned int step[51][51][51]; //棋子到达不同位置时所需要的步骤
struct str{
    char a,b,c;
}list[51*51*51]; //存储广度优先搜索算法所需要的链表
int i, j;
int n; //无向图中结点的数量
int a,b,c; //三个棋子的初始位置
int flag; //能否实现目标的标志
while(scanf("%d", &n) && n)
{
    scanf("%d%d%d", &a, &b, &c);
    memset(step,255,sizeof(step));
    int h = 0; //链表头结点的编号
    int t = 0; //链表尾结点的编号
    //链表中的第一个结点
    list[0].a = a;
    list[0].b = b;
    list[0].c = c;
    step[a][b][c] = 0;
    char arrow;
    //构造无向图中边的颜色矩阵
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
        {
            while (scanf("%c", &arrow) && !isalpha(arrow));
            g[i][j]=arrow;
        }
    flag = 0; //标志的初值
    //实现广度优先搜索
    while(h<=t) //链表没有搜索完
    {
        a = list[h].a; //取出头结点
        b = list[h].b;
        c = list[h].c;
        //判断 3 个棋子是否在同一个位置。如果是，计算结束
        if ((a==b)&&(a==c)) {flag = 1; break;}
        //头结点的步数值
        unsigned int current = step[a][b][c];
        current++; //移动 1 步
        //移动结点 a
        char bcColour = g[b][c]; //结点 b, c 的颜色
        //结点 a 与其他各个顶点相邻边的颜色，注意这是一个数组
        char *colour=g[a];
        colour++;
        //搜索每一个相邻边
    }
}

```

```

for(i=1; i<=n; i++,colour++)
    //如果不是结点 a 自身, 该边与 bc 之间的边颜色相同
    //而且存储的步数比当前所需要的步数大 (不是最优的)
    if(i!=a && *colour==bcColour && step[i][b][c]>current)
    {
        step[i][b][c]=current;          //更新步数信息
        //将该结点存储到搜索链表中
        t++;                             //更新链表的尾结点编号
        list[t].a=i;
        list[t].b=b;
        list[t].c=c;
    }
//移动结点 b, 方法同移动结点 a
bcColour = g[a][c];
colour=g[b],colour++;
for(i=1; i<=n; i++,colour++)
    if(i!=b && *colour==bcColour && step[a][i][c]>current)
    {
        step[a][i][c]=current;
        t++;
        list[t].a=a;
        list[t].b=i;
        list[t].c=c;
    }
//移动结点 c, 方法同移动结点 a
bcColour = g[a][b];
colour=g[c],colour++;
for(i=1; i<=n; i++,colour++)
    if(i!=c && *colour==bcColour && step[a][b][i]>current)
    {
        step[a][b][i]=current;
        t++;
        list[t].a = a;
        list[t].b = b;
        list[t].c = i;
    }
    h++;                                //头结点向后移动一个位置
}
//输出结果
if (flag) printf("%u\n", step[a][b][c]);
else printf("impossible\n");
}
return 0;
}

```

# ZJU1129-Erdos Numbers<sup>[1, 2, 3]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768KB

---

The Hungarian Paul Erdos (1913—1996, pronounced as "Ar-dish") was not only one of the strangest mathematicians of the 20th century, he was also among the most famous ones. He kept on publishing widely circulated papers up to a very high age, and every mathematician having the honor of being a co-author to Erdos is well respected.

Not everybody got a chance to co-author a paper with Erdos, so many people were content if they managed to publish a paper with somebody who had published a paper with Erdos. This gave rise to the so-called Erdos numbers. An author who has jointly published with Erdos had Erdos number 1. An author who had not published with Erdos but with somebody with Erdos number 1 obtained Erdos number 2, and so on. Today, nearly everybody wants to know what Erdos number he or she has. Your task is to write a program that computes Erdos numbers for a given set of scientists.

## Input

The input file contains a sequence of scenarios, each scenario consisting of a paper database and a list of names. A scenario begins with the line " $p\ n$ ", where  $p$  and  $n$  are natural numbers with  $1 \leq p \leq 32000; 1 \leq n \leq 3000$ . Following this line are  $p$  lines containing descriptions of papers (this is the paper database). A paper is described by a line of the following form:

LastName1, FirstName1, LastName2, Firstname2, ...: TitleOfThePaper

The names and the title may contain any ASCII characters between 32 and 126 except commas and colons. There will always be exactly one space character following each comma. The first name may be abbreviated, but the same name will always be written in the same way. In particular, Erdos' name is always written as "Erdos, P" ... (Umlauts like 'ö', 'ä', ... are simply written as 'o', 'a', ...)

Example:

Smith, M.N., Martin, G., Erdos, P. :Newtonian forms of prime factors matrices.

After the  $p$  papers follow  $n$  lines each containing exactly one name in the same format as in the paper database.

The line '0 0' terminates the input.

No name will consist of more than 40 characters. No line in the input file contains more than 250 characters. In each scenario there will be at most 10000 different authors.

## Output

For every scenario first print the number of the scenario in the format shown in the sample

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1129>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1391>

[3] <http://acm.uva.es/archive/nuevoportal/data/problem.php?p=2205>

output. Then print for every author name in the list of names their Erdos number based on the papers in the paper database of the scenario. The authors should be output in the order given in the input file. Authors that do not have any relation to Erdos via the papers have Erdos number infinity. Adhere to the format shown in the sample output.

Print a blank line after each case.

## Sample Input

```
2 2
Smith, M.N., Martin, G., Erdos, P.: Newtonian forms of prime factors
matrices.
Gardner, M., Martin, G.: Commuting Names
Smith, M.N.
Gardner, M.
0 0
```

## Sample Output

```
Database #1
Smith, M.N.: 1
Gardner, M.: 2
```

## Problem Source:

Mid—Central European Regional Contest 2000

### 【题目大意】

匈牙利人 Paul Erdős(1913—1996, 读音是“Ar-dish”), 不仅是一位 20 世纪最奇怪的数学家, 也是一位最著名的数学家。直到晚年, 他都持之以恒地发表广泛流传的论文。每一位数学家都把和 Erdős 一起撰写论文当作自己的荣誉。

不是每个人都有机会和 Erdős 合作写论文, 所以有许多人, 为能与 Erdős 一起合作发表过论文的人合作, 也是令人满意的。这就产生了所谓的 Erdős number。和 Erdős 共同发表论文的作者, Erdős number 是 1。虽然作者没有和 Erdős 合作发表论文, 但是和 Erdős number 是 1 的作者合作发表论文, 他的 Erdős number 是 2, 依次类推。今天, 几乎每个人都想知道, 他/她的 Erdős number 是什么。编程任务: 对给定的一组科学家, 计算每位科学家的 Erdős number。

### 输入格式

输入多组测试例, 每个测试例包含论文数据库和名字列表。测试例的第一行是自然数  $p$  和  $n$ ,  $1 \leq p \leq 32000$ ,  $1 \leq n \leq 3000$ 。接下来  $p$  行描述论文 (即论文数据库)。一篇论文一行, 格式如下:

```
LastName1, FirstName1, LastName2, Firstname2, ...: TitleOfThePaper
```

名字和标题的字符 ASCII 范围是 32~126 之间, 不包括逗号和冒号。逗号之后只有一个空格。姓可能会被缩写, 但是相同的名字总是一样的。尤其是, Erdős 的名字就写成“Erdos, P.”。(变元音字母, 像“ö”, “ä”, ...写做“o”, “a”, ...)。

示例:

```
Smith, M.N., Martin, G., Erdos, P.: Newtonian forms of prime factors matrices
```

输入  $p$  行论文之后，接下来  $n$  行是作者的名字，每行一个名字，格式同论文数据库。  
一行“00”，输入结束。

名字不超过 40 个字符。每行输入不超过 250 个字符。每个测试例中，最多有 10000 位不同的作者。

### 输出格式

对每个测试例，首先输出测试例编号，格式如样例所示。对每位作者，根据论文数据库中的作者关系，输出该作者的 Erdős number。作者的输出顺序与输入列表相同。和 Erdős 没有任何论文关系的作者，其 Erdős number 是 infinity（无穷大）。输出格式如输出样例所示。

每个测试例之后有一个空行。

### 【算法分析】

Paul Erdős 是 20 世纪一位著名的匈牙利数学家。他最大的特点就是多产，而且涉及的数学领域非常广泛。他一生和数百位同时代的数学家在不同的领域内进行过合作，发表了大约 1475 篇论文。

因为他的合作者很多，于是在数学界内，就流行起一个 Erdős number 的说法。Erdős 自己的 Erdős Number 是 0；直接和 Erdős 一起合作发表论文的作者，其 Erdős Number 就是 1。与 Erdős Number 是 1 的作者合作发表论文的作者，其 Erdős Number 就是 2，以此类推。

有很多网站介绍 Paul Erdős:

<http://topic.csdn.net/u/20070110/09/7a26d977-0fbf-4370-9385-5c667d24db88.html>  
[http://en.wikipedia.org/wiki/Paul\\_Erd%C5%91s](http://en.wikipedia.org/wiki/Paul_Erd%C5%91s)

介绍 Erdős number 的网站:

[http://en.wikipedia.org/wiki/Erd%C5%91s\\_number](http://en.wikipedia.org/wiki/Erd%C5%91s_number)  
<http://www.oakland.edu/enp/>

根据论文数据库中作者和论文的关系，以 Paul Erdős 为起点进行编号，编号的方法如前面所述，这正好符合广度优先搜索算法的策略。将作者的关系构造成邻接矩阵，就可以使用广度优先搜索算法：Paul Erdős 首先进入搜索队列，与 Paul Erdős 直接合作的所有作者进入队列，他们的 Erdős Number 是 1，然后搜索与 Erdős Number 是 1 的作者直接合作的作者，直至搜索全部作者。

本题最麻烦的事情是作者和论文的字符串处理，需要非常的细心和耐心。

#### (1) 数据结构

需要对作者的姓名进行编号。可以使用普通的字符串数组进行编号，那样比较麻烦。使用 C++ 标准模板库 STL 的 map() 容器，比较方便。姓名作为 map 的 key，编号作为 map 的 value:

```
map<string,int> name;
```

对于样例数据，其结果如表 5-2 所示。

表 5-2 样例数据的 map 结构

key	value
Erdos P.	2
Gardner M.	3
Martin G.	1
Smith M.N.	0

按读取数据的先后顺序编号，所以 Smith M.N.的编号是 0。

对一篇论文，其合作者的编号存放于数组：

```
int paper[300];
```

有了姓名编号和每篇论文的作者编号，就可以构造表示作者关系的邻接矩阵。

```
vector<int> data[10000];
```

对于样例数据，其结果如图 5-6 所示。

作者编号	与该作者合作的其他作者		
0	1	2	
1	0	2	3
2	0	1	
3	1		

图 5-6 样例数据的邻接矩阵

通过 BFS 搜索算法，得到每位作者的 Erdős Number，存放在数组：

```
int author[10000];
```

对于样例数据，其结果如表 5-3 所示。

表 5-3 作者的 Erdős Number

作者编号	0	1	2	3
Erdős Number	1	1	0	2

(2) 构造作者关系的邻接矩阵

每读取一篇论文，提取该论文的全部作者编号，存放在数组 paper 中。对作者 i，其余的作者 j (j≠i) 都是作者 i 的合作者：

```
for(i = 0; i<numPaper; i++)
    for(j = 0; j<numPaper; j++)
        if(i!=j) data[paper[i]].push_back(paper[j]);
```

变量 numPaper 是该篇论文的合作者数。

(3) 通过 BFS 搜索算法，计算每位作者的 Erdős Number

为了实现 BFS 算法，使用队列 Q：

```
struct {
    int who;                //作者编号
    int eNum;               //该作者的 Erdős Number
}Q[10000];
```

队列的首部是 Paul Erdős。

对当前作者，所有与其合作的作者，他们的 Erdős Number 都要加 1，然后将合作者加入队列，直至搜索完全部作者。由于是从 Paul Erdős 开始搜索，所以某作者进入队列越晚，他的 Erdős Number 就越大。

## 【程序代码】

程序名称: zju1129.cpp  
题 目: Erdos Numbers  
提交语言: C++  
运行时间: 170ms  
运行内存: 948KB

```
#include<iostream>
#include<vector>
#include<map>
using namespace std;

struct {
    int who;                //作者编号
    int eNum;               //该作者的 Erdős Number
}Q[10000];                 //用于 BFS 搜索算法的队列
vector<int> data[10000];    //表示作者关系的邻接矩阵
map<string,int> name;       //存放作者的姓名编号
map<string,int>::iterator pos;
int author[10000];         //每位作者的 Erdős Number

//BFS 搜索算法, 计算每位作者的 Erdős Number
void bfs(string Erdos)
{
    memset(author,-1,sizeof(author));
    memset(Q,0,sizeof(Q));
    //构造队列首部
    Q[0].who = name[Erdos];
    Q[0].eNum = 0;
    author[name[Erdos]] = 0;
    int head = 0;           //队列头指针
    int tail = 1;           //队列尾指针
    int nowAuthor, many;    //当前作者编号及合作者人数
    while(head!=tail)       //队列不为空
    {
        nowAuthor = Q[head].who;
        many = data[nowAuthor].size();    //得到合作者人数
        //与当前作者合作的作者, 还没有 Erdős Number
        for(int i = 0; i<many; i++)
            if(author[data[nowAuthor][i]]==-1)
            {
                //合作者的 Erdős Number 比作者大 1
                author[data[nowAuthor][i]] = Q[head].eNum+1;
                //合作者进入队列, 同时尾指针后移
                Q[tail].who = data[nowAuthor][i];
                Q[tail++].eNum = Q[head].eNum+1;
            }
    }
}
```



```

        }
        head++; //头指针后移
    }
}

int main()
{
    int p,n; //论文数量，待查询的作者数量
    int len;
    bool title; //遇到标题
    int numAuthor, numPaper; //作者数和论文数
    int paper[300]; //存放一篇论文的所有作者编号
    int iCase = 0; //测试例编号
    char str[300];
    string fullName;
    int i,j;
    while(scanf("%d%d",&p,&n) && (p||n)){
        for(i = 0; i<10000; i++)
            data[i].clear();
        name.clear();
        numAuthor = 0;
        //读取每一篇论文，并构造表示作者关系的邻接矩阵
        while(p--)
        {
            title = false;
            numPaper = 0;
            while(!title)
            {
                scanf("%s",str);
                len = strlen(str);
                str[len-1]=' ';
                fullName = str;
                scanf("%s",str);
                len = strlen(str);
                if(str[len-1]==':') title = true; //后面是标题
                //标题的前面也可能没有冒号，但点的后面没有逗号
                if(str[len-1]=='.') title = true;
                str[len-1]=0;
                fullName += str; //作者姓名
                pos = name.find(fullName);
                //第一次读取的作者，予以编号
                if(pos==name.end()) name[fullName] = numAuthor++;
                paper[numPaper++] = name[fullName];
                if(title) gets(str); //读取标题
            }
            //根据本篇论文中的姓名编号，将论文合作者相互加入

```

```

        for(i = 0; i<numPaper; i++)
            for(j = 0; j<numPaper; j++)
                if(i!=j) data[paper[i]].push_back(paper[j]);
    }
    // Erdos P.自己
    string Erdos = "Erdos P.";
    pos = name.find(Erdos);
    if(pos==name.end()) name[Erdos]=numAuthor++;
    //调用搜索算法，计算所有作者的 Erdős Number
    bfs(Erdos);
    printf("Database #%d\n",++iCase);
    //对每一位待查询的作者
    while(n--)
    {
        scanf("%s",str);
        printf("%s ",str);
        len = strlen(str);
        str[len-1]=' ';
        fullName = str;
        scanf("%s",str);
        printf("%s: ",str);
        fullName += str;                                //作者姓名
        pos = name.find(fullName);
        //在论文数据库中没有该作者
        if(pos==name.end()) printf("infinity\n");
        //该作者与 Paul Erdős 没有论文合作上的直接或者间接关系
        else if(author[name[fullName]]==-1) printf("infinity\n");
        //输出该作者的 Erdős Number
        else printf("%d\n",author[name[fullName]]);
    }
    printf("\n");
}
return 0;
}

```

## ZJU1136-Multiple<sup>[1、2、3]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768KB

---

a program that, given a natural number  $N$  between 0 and 4999 (inclusively), and  $M$  distinct

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1136>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1465>

[3] <http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2040>

decimal digits  $X_1, X_2, \dots, X_M$  (at least one), finds the smallest strictly positive multiple of  $N$  that has no other digits besides  $X_1, X_2, \dots, X_M$  (if such a multiple exists).

The input file has several data sets separated by an empty line, each data set having the following format:

On the first line—the number  $N$

On the second line—the number  $M$

On the following  $M$  lines—the digits  $X_1, X_2, \dots, X_M$ .

For each data set, the program should write to standard output on a single line the multiple, if such a multiple exists and 0 otherwise.

An example of input and output:

## Input

```
22
3
7
0
1

2
1
1
```

## Output

```
110
0
```

## Problem Source

Southeastern Europe 2000

### 【题目大意】

给出一个自然数  $N$  ( $0 \sim 4999$ , 包括 0 和 4999) 和  $M$  个不同的十进制数字  $X_1, X_2, \dots, X_M$  (至少一个数)。找出由数字  $X_1, X_2, \dots, X_M$  构成的正整数, 是  $N$  的最小倍数。

输入有多组测试数据, 每组数据之间有一个空行, 数据格式如下:

第一行——数字  $N$

第二行——数字  $M$

接下来  $M$  行——数字  $X_1, X_2, \dots, X_M$ 。

对每组测试数据, 假如存在此数, 则直接输出该数, 占一行; 否则输出 0。

### 【算法分析】

由数字  $X_1, X_2, \dots, X_M$  构成的正整数, 而且数字  $X_1, X_2, \dots, X_M$  是可以重复使用的, 属于可重复的全排列问题。如样例数据 1:

$X_1=7, X_2=0, X_3=1$ , 构成的正整数 110 是 22 的倍数, 而且是 22 的最小倍数。其中  $X_3$

使用了两次， $X_2$  使用了 1 次。

假设构成的数字有  $num$  位，则问题的解空间是： $m^{num}$ 。

题目中虽然没有给定  $m$  和  $num$  的大小，显然这两个数不会太大，而且  $N$  的最小倍数仍然是正整数，结合在线测试得知， $m$  和  $num$  不超过 11。

本题只要一个最优解，最适合的方法是分支限界算法。实现该算法需要用到广度优先搜索（BFS），且该算法的题目不多，就归结到搜索算法中。

关于分支限界算法，在算法类的书籍中有详细的介绍，向读者推荐参考书[1]，第 6 章，分支限界法。在Baidu知道栏目中，有“分支限界算法”词条<sup>[1]</sup>。

### （1）数据结构

设要计算的数字是  $n$ （题目中是  $N$ ），用于构造的数字的个数是  $m$ （题目中是  $M$ ）：

```
int n, m;
```

使用数组存放用于构造的数字：

```
int a[11];
```

### （2）分支限界算法的实现

分支限界（Branch and Bound）算法是一种在问题的解空间树上搜索问题的解的方法。但与回溯算法不同，分支限界算法采用广度优先或最小耗费优先的方法搜索解空间树，每一个活结点只有一次机会成为扩展结点。

利用分支限界算法对问题的解空间树进行搜索，它的搜索策略是：

- （1）产生当前扩展结点的所有子结点；
- （2）在产生的子结点中，抛弃那些不可能产生可行解（或最优解）的结点；
- （3）将其余的子结点加入活结点表；
- （4）从活结点表中选择下一个活结点作为新的扩展结点。

如此循环，直到找到问题的可行解（最优解）或活结点表为空。

从活结点表中选择下一个活结点作为新的扩展结点，根据选择方式的不同，分支限界算法通常可以分为两种形式。

（1）FIFO（First In First Out）分支限界算法：按照先进先出原则选择下一个活结点作为扩展结点，即从活结点表中取出结点的顺序与加入结点的顺序相同。

（2）最小耗费或最大收益分支限界算法：在这种情况下，每个结点都有一个耗费或收益。如果要查找一个具有最小耗费的解，那么要选择的下一个扩展结点就是活结点表中具有最小耗费的活结点；如果要查找一个具有最大收益的解，那么要选择的下一个扩展结点就是活结点表中具有最大收益的活结点。

本题采用 FIFO 分支限界算法。

### ● 数据结构

由于  $N$  的最小倍数是由数字  $X_1, X_2, \dots, X_M$  组合而来的，假设位数是  $num$ ，而数字存储到数组中：

```
int y[11];
```

存放活结点，需要使用队列。可以使用 C++ 标准模板库函数 `queue()`，这里使用数组实现：

```
int q[5001][3];
```

每个结点包含下列 3 个元素：

---

[1] <http://baike.baidu.com/view/1304851.htm>

- ①变量  $x$ : 当前结点时, 构造的数字除以  $n$  的余数 (即模  $n$ );
- ②当前结点时, 使用到的原始数据;
- ③变量  $head$ : 队列的头指针, 下一个要构造的数的前缀。

队列的头指针和尾指针:

```
int head = 0;
int tail = 0;
```

辅助数组  $b$ , 表示下标为  $x$  的值是否在队列中:

```
char b[5001];
```

初值为 1, 表示  $x$  不在队列中; 否则为 0。

● 算法的实现

当队列不为空时, 将数字  $X_1, X_2, \dots, X_M$  逐一取出, 在当前活结点的尾部增加 1 位数字:

```
x = (q[head][0] * 10 + a[i]) % n;
```

如果  $b[x]=1$ , 表示该  $x$  不在队列中, 随即加入队列:

```
tail++;
q[tail][0] = x;
q[tail][1] = a[i];
q[tail][2] = head;
```

并记  $b[x]=0$ 。

如果  $x=0$ ; 表示找到了  $n$  的倍数 (下面会讨论这就是最小倍数), 将构造的数字放到数组  $y$  中 (根据前缀循环构造), 并输出。

(3) 实现  $N$  的最小倍数

为了确保找到的就是  $N$  的最小倍数, 我们每次从数字  $X_1, X_2, \dots, X_M$  中, 首先取出最小的数字构造, 然后是次小的, 等等, 这样第一个满足条件的数字就是最小的倍数。通过对数字  $X_1, X_2, \dots, X_M$  按升序排序, 就可以实现。

【程序代码】

---

程序名称:	zju1136.c
题 目:	Multiple
提交语言:	C
运行时间:	30ms
运行内存:	160KB

---

```
#include <stdio.h>
#include <memory.h>
#include <stdlib.h>
```

```
//比较因子
int cmp(const void * a,const void * b)
{
    return (*(int *)a - *(int *)b);
}
```

```
int n, m;
```

```
//要计算的数字, 用于构造的数字的个数
```

```

int a[11];                                //用于构造的数字

//分支限界算法的实现
void bfs()
{
    int i, j, k;
    int x;
    int num;                                //构成数字的位数
    int y[11];                            //构成的数字
    char b[5001];                          //扩展结点是否在队列中
    int q[5001][3];                        //队列
    q[0][0] = q[0][1] = q[0][2] = 0;
    int head = 0;                          //队列的头指针
    int tail = 0;                          //队列的尾指针
    memset(b, 1, sizeof(b));
    do                                    //扩展所有的结点
    {
        for (i=0; i<m; i++)                //生成所有的子结点
        {
            //数字 0 不能作为数字的第一位
            if (head == 0 && a[i] == 0) continue;
            //扩展结点
            x = (q[head][0] * 10 + a[i]) % n;
            if (b[x])                        //该结点不在队列中
            {
                tail++;                    //加入队列
                q[tail][0] = x;
                q[tail][1] = a[i];
                q[tail][2] = head;
                b[x] = 0;                    //修改标记
                if (x==0)                    //找到了
                {
                    num = 0;
                    k = tail;                //最后一位数字的结点编号
                    while (1)
                    {
                        num++;
                        //取出构造的数字，放到数组 y 中
                        y[num] = q[k][1];
                        k = q[k][2];          //取出前缀
                        if (k == 0)break;      //没有了
                    }
                    //输出结果
                    for (j = num; j >= 1; j--)
                        printf("%d", y[j]);
                    printf("\n");
                }
            }
        }
    }
}

```

```

        return;
    }
}

    head++;
} while (head <= tail);
printf("0\n");
}

int main()
{
    int i;
    while(scanf("%d %d", &n, &m) != EOF)
    {
        for (i = 0; i < m; i++)
            scanf("%d", &a[i]);
        qsort(a, m, sizeof(int), cmp);
        if (n == 0) printf("0\n");
        else bfs();
    }
    return 0;
}

```

//头结点后移  
//队列不空

//读取原始数据  
//按升序排序

## ZJU1142-Maze<sup>[1、2、3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

By filling a rectangle with slashes (/) and backslashes (\), you can generate nice little mazes. Here is an example:

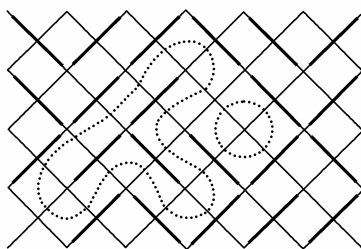


Figure 5-7 Maze

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1142>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1103>

[3] <http://acm.uva.es/p/v7/705.html> (Slash Maze)

As you can see, paths in the maze cannot branch, so the whole maze only contains cyclic paths and paths entering somewhere and leaving somewhere else. We are only interested in the cycles. In our example, there are two of them.

Your task is to write a program that counts the cycles and finds the length of the longest one. The length is defined as the number of small squares the cycle consists of (the ones bordered by gray lines in the picture). In this example, the long cycle has length 16 and the short one length 4.

## Input

The input contains several maze descriptions. Each description begins with one line containing two integers  $w$  and  $h$  ( $1 \leq w, h \leq 75$ ), the width and the height of the maze. The next  $h$  lines represent the maze itself, and contain  $w$  characters each; all these characters will be either "/" or "\".

The input is terminated by a test case beginning with  $w=h=0$ . This case should not be processed.

## Output

For each maze, first output the line "Maze #n:", where  $n$  is the number of the maze. Then, output the line "kCycles; the longest has length l.", where  $k$  is the number of cycles in the maze and  $l$  the length of the longest of the cycles. If the maze does not contain any cycles, output the line "There are no cycles."

Output a blank line after each test case.

## Sample Input

```
6 4
\\//\\//
\\///\\//
//\\\\//
\\//\\//
3 3
///
\\//
\\\\
0 0
```

## Sample Output

```
Maze #1:
2 Cycles; the longest has length 16.
Maze #2:
There are no cycles.
```

## Problem Source

Mid—Central European Regional Contest 1999



## 【题目大意】

用斜杆(/)和反斜杆(\)处理一个长方形，你能得到一个小迷宫。

如图 5-7 所示，迷宫中的路线不能产生分支。因此，整个迷宫只有循环路线和进入与离开某个地方的路线。我们只关注循环路线。本例中，有两条循环路线。

程序搜索迷宫中的循环路线和计算其中最长的路线长度。长度定义为循环路线经过的格子数。本例中，最长路线的长度 16，最短路线的长度 4。

### 输入格式

输入有多组迷宫。对每组迷宫，第一行是两个整数  $w$  和  $h$  ( $1 \leq w, h \leq 75$ )，表示迷宫的宽度和高度。接下来的  $h$  行表示迷宫，每行  $w$  个字符，字符只是 “/” 或 “\”。

输入  $w=h=0$  结束，这组数据不需处理。

### 输出格式

对于每个迷宫，先输出一行 “Maze #n:”， $n$  是迷宫的编号。然后，输出 “k Cycles; the longest has length l.”， $k$  是迷宫里循环路线的数量， $l$  是所有循环路线中的最长长度。如果迷宫中没有循环路线，输出 “There are no cycles.”。

每组测试数据后输出一空行。

## 【算法分析】

迷宫类的题目，一般都是搜索，本题是使用深度优先搜索算法实现的。如果把循环的路线看成一棵树，则所有循环的路线构成一个森林，可以采用并查集算法，有兴趣的读者可以参考光盘上的代码：zju1142—Union.cpp。

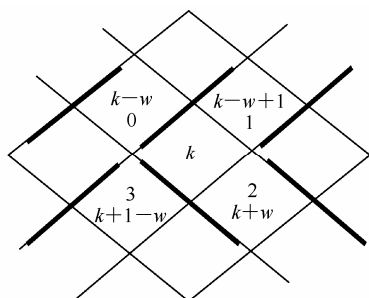
### (1) 数据结构

存储迷宫的数组：

```
char maze[N][N];
```

迷宫的高度和宽度：

```
int h, w;
```



错误!

图 5-7 迷宫的数据结构

迷宫中的方格个数：

```
int n = (h-1) * (w+w-1) + w - 1;
```

### (2) 建立迷宫

使用下面两个数组描述迷宫：

迷宫中方格的墙，有墙时为 0，没有墙时为 1：

```
char wall[N*N][4];
```

迷宫中某个方格的相邻方格编号：

```
int next[N*N][4];
```

如图 5-8 所示：

图中，对应样例数据 1 的第 4 行第 2 个单元， $k=19$ 。

从图中看出：

```
wall[19][0]=0, wall[19][1]=1, wall[19][2]=1, wall[19][3]=0;
```

```
next[19][0]=13, next[19][1]=14, next[19][2]=25, next[19][3]=24
```

注意：对偶数行，只有  $w-1$  个方格。

### (3) 深度优先搜索，获得循环路线的长度

迷宫构建成功以后，就可以搜索了。为了避免对同一个单元重复搜索，采用数组做标记：

```
char used[N*N];
```

数组的初值为 0。搜索某个方格之后，就标记为 1。

搜索由函数完成实现：

```
char dfs(int i,int length,int start)
```

其中形参  $i$  为搜索的方格， $length$  是该方格时循环路线的长度， $start$  是本次搜索的起始点。

显然当  $length > 2$ ， $i = start$  时，就是找到了一条循环路线；否则没有循环路线或者有分支路线 ( $used[i] = 1$ )。

搜索时，沿着当前方格  $i$  向 4 个方向延伸（如果没有墙的话）。

## 【程序代码】

---

程序名称： zju1142.c

题 目： Maze

提交语言： C

运行时间： 0ms

运行内存： 784KB

---

```
#include<stdio.h>
#include<string.h>

#define N 160
char wall[N*N][4];           //迷宫中方格的墙
char used[N*N];              //节点是否已经访问
int next[N*N][4];            //迷宫中某个方格的相邻方格编号
int path[N*N];               //迷宫中某个方格的路线长度
int n;                        //迷宫中的方格个数
int h, w;                     //迷宫的高和宽

//构建迷宫
void side(char maze[][N]) {
    memset(wall,0,sizeof(wall));
    int k = 0;                //迷宫中方格的编号
    int i, j;
    for(i = 0; i<2*h-1; i++)
        for(j = 0; j<w; j++,k++)
        {
            //偶数行，少一个方格，即只有 w-1 个方格
            if(i%2==0 && j==w-1) break;
            if(i==0)           //第 0 行
            {
                if (maze[0][j]=='/') wall[k][3] = 1;
                if (maze[0][j+1]!='/') wall[k][2] = 1;
                next[k][3] = w-1+j;
                next[k][2] = w+j;
            }
            else if(i==2*h-2)   //最后一行
```

```

    {
        if (maze[h-1][j]!='/') wall[k][0] = 1;
        if (maze[h-1][j+1]=='/') wall[k][1] = 1;
        next[k][0] = n-(w+w-1)+j;
        next[k][1] = n-(w+w-1)+j+1;
    }
else
{
    if(i%2==0)                //偶数行
    {
        if (maze[i/2][j]=='/') wall[k][3] = 1;
        else wall[k][0] = 1;
        if (maze[i/2][j+1]!='/') wall[k][2] = 1;
        else wall[k][1] = 1;
        next[k][0] = k-w;
        next[k][1] = k-w+1;
        next[k][2] = k+w;
        next[k][3] = k+w-1;
    }
    else                //奇数行
    {
        if (maze[i/2][j]=='/') wall[k][1] = 1;
        else wall[k][0]=1;
        if (maze[i/2+1][j]=='/') wall[k][3] = 1;
        else wall[k][2]=1;
        next[k][0] = k-w;
        next[k][1] = k-w+1;
        next[k][2] = k+w;
        next[k][3] = k+w-1;
    }
}
//奇数行的第一个方格
if (j==0 && i%2) wall[k][0] = wall[k][3] = 0;
//奇数行的最后一个方格
if (i%2 && j==w-1)
    wall[k][2] = wall[k][1] = 0;
}
}

```

//深度优先搜索，获得循环路线的长度，并将已经访问过的结点做标记

```

char dfs(int i,int length,int start)
{
    path[i] = length;
    //搜索到起始结点，是一条循环路线
    if (i==start && length>2) return 1;
    //不是循环路线，有分支路线

```

```

    if (used[i]) return 0;
    //标记为访问过
    used[i]=1;
    //如果没有墙的话, 进入下一个方格
    if (wall[i][0] && dfs(next[i][0],length+1,start)) return 1;
    if (wall[i][1] && dfs(next[i][1],length+1,start)) return 1;
    if (wall[i][2] && dfs(next[i][2],length+1,start)) return 1;
    if (wall[i][3] && dfs(next[i][3],length+1,start)) return 1;
    return 0; //不封闭的路线
}

int main()
{
    int i;
    int iCase = 1; //测试例数
    char maze[N][N]; //迷宫原始数据
    while(scanf("%d %d",&w,&h) && w)
    {
        for(i=0; i<h; i++) //读取迷宫
            scanf("%s",maze[i]);
        n = (h-1)*(w+w-1)+w-1; //计算方格数
        //构建迷宫
        side(maze);

        //逐个方格查询循环路线的情况
        memset(used, 0, sizeof(used));
        int total = 0; //记录循环路线的数量
        int max = 0; //记录循环路线中最长的长度
        memset(path, 0, sizeof(path));
        for(i=0; i<n; i++)
            //如果该方格没有被标记为访问过
            if(!used[i])
            {
                if (dfs(i,0,i)) //找到一条循环路线
                {
                    if (path[i]>max) max = path[i];
                    total++;
                }
                used[i]=1;
            }
        //输出结果
        printf("Maze #%d:\n",iCase++);
        if(total>0)
            printf("%d Cycles; the longest has length %d.\n",total,max);
        else
            printf("There are no cycles.\n");
    }
}

```

```

        printf("\n");
    }
    return 0;
}

```

## ZJU1148-The Game<sup>[1, 2, 3]</sup>

Time Limit: 1 Second

Memory Limit: 32768KB

One morning, you wake up and think: "I am such a good programmer. Why not make some money?" So you decide to write a computer game.

The game takes place on a rectangular board consisting of  $w \times h$  squares. Each square might or might not contain a game piece, as shown in the picture.

One important aspect of the game is whether two game pieces can be connected by a path which satisfies the two following properties:

It consists of straight segments, each one being either horizontal or vertical.

It does not cross any other game pieces.

(It is allowed that the path leaves the board temporarily.)

Here is an example:

The game pieces at (1,3) and at (4, 4) can be connected. The game pieces at (2, 3) and (3, 4) cannot be connected; each path would cross at least one other game piece.

The part of the game you have to write now is the one testing whether two game pieces can be connected according to the rules above.

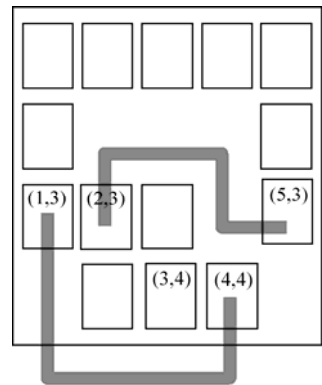


Figure 5-9 The board

## Input

The input contains descriptions of several different game situations. The first line of each description contains two integers  $w$  and  $h$  ( $1 \leq w, h \leq 75$ ), the width and the height of the board. The next  $h$  lines describe the contents of the board; each of these lines contains exactly  $w$  characters: a "X" if there is a game piece at this location, and a space if there is no game piece.

Each description is followed by several lines containing four integers  $x_1, y_1, x_2, y_2$  each satisfying  $1 \leq x_1, x_2 \leq w, 1 \leq y_1, y_2 \leq h$ . These are the coordinates of two game pieces. (The upper left corner has the coordinates (1, 1).) These two game pieces will always be different. The list of pairs of game pieces for a board will be terminated by a line containing "0 0 0 0".

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1148>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1101>

[3] <http://acm.uva.es/p/v7/710.html>

The entire input is terminated by a test case starting with  $w=h=0$ . This test case should not be processed.

## Output

For each board, output the line "Board #n:", where n is the number of the board. Then, output one line for each pair of game pieces associated with the board description. Each of these lines has to start with "Pair m: ", where m is the number of the pair (starting the count with 1 for each board). Follow this by "k segments.", where k is the minimum number of segments for a path connecting the two game pieces, or "impossible.", if it is not possible to connect the two game pieces as described above.

Output a blank line after each board.

## Sample Input

```
5 4
XXXXX
X  X
XXX X
XXX
2 3 5 3
1 3 4 4
2 3 3 4
0 0 0 0
0 0
```

## Sample Output

```
Board #1:
Pair 1: 4 segments.
Pair 2: 3 segments.
Pair 3: impossible.
```

## Problem Source

Mid—Central European Regional Contest 1999

### 【题目大意】

一天早晨，你醒后想到：“我是一个优秀的程序设计者。为什么不赚一些钱呢？”因此你决定编写一个计算机游戏。

游戏在一个有  $w \times h$  个方格的矩形板上进行。每个方格也许有或也许没有游戏板，如图 5-9 所示。

该游戏有一条重要的规则：用一条路径把两个游戏板连接起来，且满足以下两个要求：

- (1) 路径都是直线段，每段路径要么是水平的，要么是垂直的。
- (2) 路径不允许跨越任何游戏板。（允许路径暂时离开游戏板。）

如图 5-9 所示例题：

游戏板 (1, 3) 和 (4, 4) 能连接起来；而游戏板 (2, 3) 和 (3, 4) 不能连接起来，因为无论哪条路径，都要跨越其他游戏板。

根据游戏规则，程序判断其中的两块游戏板是否能连接在一起。

### 输入格式

输入有多组不同的游戏模式。对每组游戏模式，第 1 行是两个整数  $w$  和  $h$  ( $1 \leq w, h \leq 75$ )，是矩形板的宽度和高度。接下来  $h$  行表示矩形板的构成：每行有  $w$  个字符：“X”表示此处有游戏板，空格表示此处没有游戏板。

然后是几行数据，每行 4 个整数  $x_1, y_1, x_2, y_2$ ， $1 \leq x_1, x_2 \leq w, 1 \leq y_1, y_2 \leq h$ ，表示两块游戏板的坐标。（左上角的坐标是 (1, 1)。）任何两块游戏板的坐标都是不同的。当两块游戏板的坐标是 “0 0 0 0” 时，表示输入数据结束。

输入  $w=h=0$ ，程序结束。不需要处理此行。

### 输出格式

对每个游戏模式，输出一行 “Board #n:”， $n$  是游戏模式的编号。然后，相应于输入中的每对游戏板输出一行：每行开始是 “Pair m:”， $m$  是每对游戏板的编号（每个游戏板的开始都是 1。）然后是 “k segments.”， $k$  是连接这对游戏板的最小步数；如果不可能，输出 “impossible.”。

每个游戏模式后输出一个空行。

### 【算法分析】

看起来像连连看游戏，但连连看游戏只能有两个拐弯，而这个游戏允许有多个拐弯。很容易想到广度优先搜索，只是这种游戏在连的时候，要尽可能少的拐弯。这在搜索时沿着一个方向，只要能够连的话，就一直连下去即可。

#### （1）数据结构和数据的读取

使用数组表示矩形游戏板：

```
char g[80][80];
```

由于连接时，可以暂时离开矩形游戏板，所以矩形游戏板的四周要加大一圈。对样例数据，如图 5-10 所示。

	X	X	X	X	X	
	X				X	
	X	X	X		X	
		X	X	X		

图 5-10 矩形游戏板的数据结构

读取矩形游戏板的数据时，每次读取一行，按图示位置存放，然后四周补齐空格。

#### （2）广度优先搜索（BFS）算法的实现

##### ● 数据结构

用于存储广度优先搜索（BFS）的队列：

```
struct{
    int x, y;
} q[3000];
```

存储从起点开始的步数:

```
int steps[80][80];
```

需要连接的一对游戏板的坐标:

```
int sx, sy, dx, dy;
```

### ● 与普通广度优先搜索算法的区别

一般的广度优先搜索算法,只需要向结点的四周搜索就行了,但是本题像连连看游戏,需要尽可能少的拐弯。这就要求在搜索时,在同一个方向,应该一直往前走。为了做到这一点,只要将当前的坐标增量继续往下加就可以了,一直到不能前进为止。

算法的具体实现,请读者参考代码中的注释。

## 【程序代码】

---

程序名称:	zju1148.c
题    目:	The Game
提交语言:	C
运行时间:	0ms
运行内存:	236KB

---

```
#include <stdio.h>
#include <memory.h>

//用于广度优先搜索(BFS)的队列
struct{
    int x, y;
} q[3000];

char g[80][80]; //存储矩形游戏板
int steps[80][80]; //存储 BFS 算法中,从起点开始的步数
//BFS 算法的搜索方向
int dir[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

int main(){
    int i;
    int n = 1; //游戏模式的编号
    int sx, sy, dx, dy; //一对游戏板的坐标
    int h, w; //矩形板的宽度和高度

    while(scanf("%d%d", &w, &h) && (h || w)){
        getchar(); //读取回车
        int m = 1; //查询的每对游戏板的编号
        for(i = 1; i <= h; i++) //读取游戏板
            gets(&g[i][1]);
        //矩形游戏板的四周加空格
        for(i = 0; i < w + 2; i++)
            g[0][i] = g[h + 1][i] = ' ';
        for(i = 1; i <= h; i++)
```



```

        g[i][0] = g[i][w + 1] = ' ';

//查询每对游戏板
printf("Board #d:\n", n++);
while(scanf("%d%d%d", &sy, &sx, &dy, &dx)){
    if(!(sx || sy || dx || dy)) break;
    memset(steps, -1, sizeof(steps));
    g[sx][sy] = g[dx][dy] = ' ';           //该对游戏板置空格, 便于查询
    steps[sx][sy] = 0;
    int top = 0;                           //头结点指针
    int tail = 0;                          //尾结点指针
    int x0, y0;                            //当前结点
    q[top].x = sx;                         //头结点
    q[top].y = sy;
    //队列不为空, 且该结点没有搜索过
    while(top <= tail && steps[dx][dy] == -1){
        //向4个方向搜索
        for(i = 0; i < 4; i++){
            x0 = q[top].x + dir[i][0];
            y0 = q[top].y + dir[i][1];
            //沿着该方向一直搜索
            while(x0 >= 0 && x0 <= h+1 && y0 >= 0 && y0 <= w+1
                && g[x0][y0] == ' ' && steps[x0][y0] == -1){
                //当前结点加入到队列尾部
                tail++;
                q[tail].x = x0;
                q[tail].y = y0;
                //步数加1
                steps[x0][y0] = steps[q[top].x][q[top].y] + 1;
                //该方向的下一个结点
                x0 += dir[i][0];
                y0 += dir[i][1];
            }
        }
        top++;
    }
    g[sx][sy] = g[dx][dy] = 'X';           //恢复该对游戏板的原先状态
    if (steps[dx][dy] == -1)
        printf("Pair %d: impossible.\n", m++);
    else
        printf("Pair %d: %d segments.\n", m++, steps[dx][dy]);
}
printf("\n");
}
return 0;
}

```

## 第六章 动态规划算法题

在本章的题目主要是动态规划算法题。需要使用动态规划的方法，实现题目的要求。

### ZJU1093-Monkey and Banana<sup>[1、2、3]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768K

---

A group of researchers are designing an experiment to test the IQ of a monkey. They will hang a banana at the roof of a building, and at the mean time, provide the monkey with some blocks. If the monkey is clever enough, it shall be able to reach the banana by placing one block on the top another to build a tower and climb up to get its favorite food.

The researchers have  $n$  types of blocks, and an unlimited supply of blocks of each type. Each type- $i$  block was a rectangular solid with linear dimensions  $(x_i, y_i, z_i)$ . A block could be reoriented so that any two of its three dimensions determined the dimensions of the base and the other dimension was the height.

They want to make sure that the tallest tower possible by stacking blocks can reach the roof. The problem is that, in building a tower, one block could only be placed on top of another block as long as the two base dimensions of the upper block were both strictly smaller than the corresponding base dimensions of the lower block because there has to be some space for the monkey to step on. This meant, for example, that blocks oriented to have equal-sized bases couldn't be stacked.

Your job is to write a program that determines the height of the tallest tower the monkey can build with a given set of blocks.

#### Input Specification

The input file will contain one or more test cases. The first line of each test case contains an integer  $n$ , representing the number of different blocks in the following data set. The maximum value for  $n$  is 30.

Each of the next  $n$  lines contains three integers representing the values  $x_i$ ,  $y_i$  and  $z_i$ .

Input is terminated by a value of zero (0) for  $n$ .

#### Output Specification

For each test case, print one line containing the case number (they are numbered sequentially

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1093>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2241>

[3] <http://acm.uva.es/p/v4/437.html>

starting from 1) and the height of the tallest possible tower in the format "Case case: maximum height = height"

## Sample Input

1	7	5
10 20 30	1 1 1	31 41 59
2	2 2 2	26 53 58
6 8 10	3 3 3	97 93 23
5 5 5	4 4 4	84 62 64
	5 5 5	33 83 27
	6 6 6	0
	7 7 7	

## Sample Output

```
Case 1: maximum height = 40
Case 2: maximum height = 21
Case 3: maximum height = 28
Case 4: maximum height = 342
```

## Problem Source

University of Ulm Local Contest 1996

### 【题目大意】

一组研究人员正在设计一个测试猴子 IQ 的实验。他们把香蕉吊在屋顶上，同时给猴子提供了砖块。如果猴子够聪明，它会把砖块一个个叠起来做成一个塔，然后爬上去拿到自己喜爱的食物。

研究人员有  $n$  种不同的砖块，而且每种砖块都是取之不尽的。每种块砖都是长方体，第  $i$  种砖块的大小是  $(x_i, y_i, z_i)$ 。砖块能够翻转，可以将任意两边当作底面，剩下的那边作为高。

他们想确定用砖块搭成的最高塔，能否帮助猴子够着屋顶。问题是，在叠塔过程中，要放的那块砖，其底面两条边都要小于下面那块砖的两条边，这是为了留个空间给猴子踩脚。例如，底面相同尺寸的砖块不能相叠。

编程任务：对于给定的砖块，计算猴子能够叠塔的最大高度。

### 输入格式

输入包含一组或多组测试例。每组测试例的第一行是一个整数  $n$ ，表示砖块的种类数。 $n$  的最大值是 30。

接着  $n$  行，每行三个整数： $x_i, y_i$  和  $z_i$ 。

当  $n$  为零时，表示输入结束。

### 输出格式

对每组测试例，输出一行：测试例编号 case(从 1 开始编号)，塔能够达到的最大高度 height。输出格式为：“Case case: maximum height = height”。

【算法分析】

在University of Ulm Local Contest 1996 的竞赛资料页面<sup>[1]</sup>上，给出的标程有两个算法：DP 算法和Floyd算法。这里只介绍动态规划（DP）算法的实现。

（1）砖块的表示

由于砖块可以任意摆放，所以一块砖有 6 种不同的摆放方式,见表 6-1。

表 6-1 块砖的不同摆放方式

摆放方式	底面	高	说明
1	$x, y$	$z$	直放、横放各一次
2	$y, x$	$z$	
3	$y, z$	$x$	直放、横放各一次
4	$z, y$	$x$	
5	$x, z$	$y$	直放、横放各一次
6	$z, x$	$y$	

这是一个三维问题，处理起来肯定麻烦。从表 6-1 看出，将一块砖的不同摆放方式，转化为 6 种不同的砖块，按同一方式摆放，这就将三维问题转化为一维问题，用数组表示：

```
int box[100][3];
```

构造数组时，使用了函数：

```
void oriente(int serial, int a, int b, int c)
```

使语句更简洁。形参 serial 是数组的当前下标，( $a, b$ ) 是底面， $c$  是高。这里同一底面的砖块只构造了一次，在 DP 时直放和横放都判断一下就可以了。

（2）动态规划算法的实现

在数组 box 的第 0 个单元，增加一块砖：三个方向都是 $\infty$ ，表示地面。

将每一块砖实现的高度保存到数组：

```
int height[100];
```

动态规划算法由函数实现：

```
int DP(int serial)
```

最优子结构：

对第 serial 块砖，如果能够叠放到塔中第  $i$  块 ( $i1 \sim \text{number}$ ) 砖上面，则其高度  $t$  就是前面  $i$  块砖获得的最优高度加上本块砖的高度，并且放上这块砖确实能够得到最优解 ( $t > \text{height}[\text{serial}]$ )。因为一块小的砖放在塔的上部比放在塔的下部能够获得更好的解。

【程序代码】

程序名称：	zju1093-DP.c
题    目：	Monkey and Banana
提交语言：	C
运行时间：	00:00.00
运行内存：	392K

```
#include <stdio.h>
```

[1] <http://www.informatik.uni-ulm.de/acm/Locals/1996/>

```

#define MAX 9999999

int box[100][3];           //砖块
int height[100];          //各种砖块能够达到的高度
int number;                //转换后的砖块种类数, 是  $3 \times n + 1$  (地板)

//构造砖块数组
//形参 serial 是数组的当前下标, (a, b) 是底面, c 是高
void oriente(int serial, int a, int b, int c) {
    box[serial][0] = a;
    box[serial][1] = b;
    box[serial][2] = c;
}

//动态规划算法的实现
int DP(int serial){
    //第 serial 块砖已经计算过
    if (height[serial] != -1) return height[serial];
    //所有砖块计算完毕
    if (serial > number) return 0;
    //搜索每一块砖
    int t = 0;
    int i;
    for (i = 1; i <= number; ++i) {
        //第 serial 块砖能够放上去
        if ((box[serial][0] > box[i][0] && box[serial][1] > box[i][1]) ||
            (box[serial][0] > box[i][1] && box[serial][1] > box[i][0]))
            t = DP(i) + box[i][2];
        //能够获得最优解
        if (t > height[serial]) height[serial] = t;
    }
    return height[serial];
}

int main(){
    int n;                //砖块的种类数
    int cases = 0;        //测试例数
    int a, b, c;
    while (scanf("%d", &n) && n){
        //增加一块地板
        box[0][0] = box[0][1] = box[0][2] = MAX;
        //读取砖块数据, 构造砖块数组
        number = 0;
        int i;
        for (i = 0; i < n; ++i){

```

```

        scanf("%d%d%d", &a, &b, &c);
        oriente(++number, a, b, c);
        oriente(++number, b, c, a);
        oriente(++number, c, a, b);
    }
    //表示高度的数组初始化
    for (i = 0; i <= number; ++i)
        height[i] = -1;
    //输出结果
    printf("Case %d: maximum height = %d\n", ++cases, DP(0));
}
return 0;
}

```

### 【其他算法】

如果把一块砖的所有 6 种摆放方式，转化为 6 种不同的砖块，相当于有  $6n$  种砖块，然后按一个方向从大到小排序，就像叠罗汉一样将它们全部叠起来，再依次检查每一块与其下面的所有砖块是否满足摆放条件，将每一块砖放到塔中能够获得的的最大高度记录到数组中：

```
int height[200];
```

再从数组 **height** 中查找最大值，该最大值就是答案 **answer**。

该方法比前一个方法更直观，所以 University of Ulm Local Contest 1996 的标程中，就是使用了这种方法。

### 【程序代码】

---

程序名称:	zju1093-sort-DP.c
题 目:	Monkey and Banana
提交语言:	C++
运行时间:	00:00.01
运行内存:	448K

---

```

#include <stdio.h>
#include <stdlib.h>

```

```
//表示砖块的数据结构
```

```
struct block
```

```
{
```

```
    int x, y, z;
```

```
}box[200]; //存放砖块的数组
```

```
//构造砖块数组
```

```
//形参 k 是数组的当前下标, (x, y) 是底面, z 是高
```

```
void oriente(int k, int x, int y, int z) {
```

```
    box[k].x = x;
```

```
    box[k].y = y;
```

```
    box[k].z = z;
```

```

}

//比较算法, 按 x 方向降序排序
int cmp(const void * a,const void * b)
{
    return ((block*)b)->x-((block*)a)->x;
}

int main ()
{
    int i,j;
    int n;                //砖块的种类数
    int x,y,z;
    int height[200];      //各种砖块能够达到的高度
    int Case=1;           //测试例数
    while(scanf("%d", &n) && n)
    {
        int number = 0;
        for(j=0; j<n; j++)
        {
            scanf("%d%d%d", &x, &y, &z);
            oriente(number++, x, y, z);
            oriente(number++, x, z, y);
            oriente(number++, y, z, x);
            oriente(number++, y, x, z);
            oriente(number++, z, x, y);
            oriente(number++, z, y, x);
        }
        n *= 6;           //砖块数增长到 6 倍
        //将所有砖块排序
        qsort(box, n, sizeof(block), cmp);
        //每块砖本身的高度
        for(i=0; i<n; i++)
            height[i] = box[i].z;
        int max;
        int answer = 0;    //答案
        for(i=0; i<n; i++)
        {
            //对每一块砖 i, 计算将其放到塔中所能获得的最大高度
            max=0;
            for(j=i-1; j>=0; j--)
                if(box[i].x<box[j].x && box[i].y<box[j].y &&
                    max<height[j])
                    max = height[j];
        }
    }
}

```

```

        //加上本身的高度
        height[i] += max;
        //查找数组 height 的最大值
        if(answer<height[i]) answer = height[i];
    }
    printf("Case %d: maximum height = %d\n", Case++, answer);
}
return 0;
}

```

## ZJU1100-Mondriaan's Dream<sup>[1、2、3]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768K

---

Squares and rectangles fascinated the famous Dutch painter Piet Mondriaan. One night, after producing the drawings in his 'toilet series' (where he had to use his toilet paper to draw on, for all of his paper was filled with squares and rectangles), he dreamt of filling a large rectangle with small rectangles of width 2 and height 1 in varying ways.

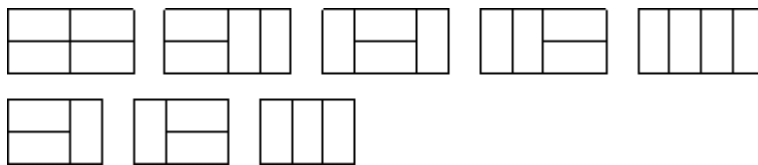


Figure 6-1

Expert as he was in this material, he saw at a glance that he'll need a computer to calculate the number of ways to fill the large rectangle whose dimensions were integer values, as well. Help him, so that his dream won't turn into a nightmare!

### Input Specification

The input file contains several test cases. Each test case is made up of two integer numbers: the height  $h$  and the width  $w$  of the large rectangle. Input is terminated by  $h=w=0$ . Otherwise,  $1 \leq h, w \leq 11$ .

### Output Specification

For each test case, output the number of different ways the given rectangle can be filled with small rectangles of size 2 times 1. Assume the given large rectangle is oriented, i.e. count symmetrical tilings multiple times.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1100>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2411>

[3] <http://www.informatik.uni-ulm.de/acm/Locals/2000/html/dream.html>



## Sample Input

```
1 2
1 3
1 4
2 2
2 3
2 4
2 11
4 11
0 0
```

## Sample Output

```
1
0
1
2
3
5
144
51205
```

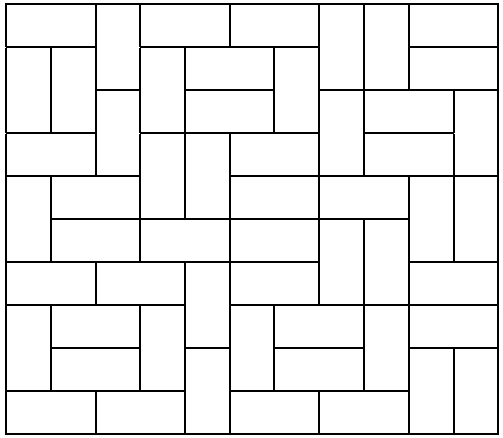


Figure 6-2

## Problem Source

University of Ulm Local Contest 2000

### 【翻译】

著名的荷兰画家 **Piet Mondriaan** 对正方形和长方形特别着迷。一天晚上，在完成他的“厕所系列”作品（在厕所里，他只能用卫生纸作画，他把所有的卫生纸都画上了正方形和长方形）后，他想用宽为 2 高为 1 的小矩形，以多种方法填充一个大的矩形如图 6-1 所示。

尽管他是这方面的专家，一眼就知道，他需要一台计算机来计算填充一个大的矩形时，到底有多少种方法。大矩形的边长是整数。请帮助他梦想成真！

### 输入格式

输入有多组测试例。每组测试例有两个整数：大矩形的高  $h$  和宽  $w$  ( $1 \leq h, w \leq 11$ )。输入  $h=w=0$  时结束。

### 输出格式

对每组测试例，用  $2 \times 1$  的小矩形填充给定的矩形时，输出不同的方法数。假设大长方形是有方向性的，即对称的填充可以重复计数（如图 6-2 中  $2 \times 3$  矩形的前两个填充）。

### 【算法分析】

本题是深度优先搜索和动态规划的结合应用，算法参考自标程<sup>[1]</sup>。为表达方便，我们将  $2 \times 1$  的小矩形简称为“砖块”。

[1] <http://www.informatik.uni-ulm.de/acm/Locals/2000/solution/>

(1) 采用深度优先搜索算法，计算第一层的填充状态

把每一列的方格压缩为二进制编码，搜索上一列到当前列的状态转化是否能够达到。对于每一个位置，我们有三种放置方法：

- ① 竖直放置砖块
- ② 水平放置砖块
- ③ 不放置砖块

对每个方格，1 表示填充砖块，0 表示没有填充砖块。将一层的状态可以存放在一个二进制数中，计算时不必进行二进制转换，直接使用位运算。

该算法由下列函数实现：

```
void dfs (int n, int from, int to)
```

表示当前从左往右数第  $n$  个方格， $from$  表示前  $n$  个方格在这一层的编码， $to$  表示下一层的编码。显然  $from$  和  $to$  的二进制长度与大长方形的宽度  $W$  是一致的。

状态转化的数量  $nTran$  与宽度  $W$  是密切相关的，在标程中给出了一个计算公式：

$$nTran = \left[ \left( \sqrt{2} + 1 \right)^W - \left( \sqrt{2} - 1 \right)^W \right] \times \left( \sqrt{2} + 2 \right) / 4$$

计算结果如表 6-2 所示。

表 6-2 不同宽度下状态转化的数量

W	2	3	4	5	6	7	8	9	10	11
nTran	5	12	29	70	169	408	985	2378	5741	13860

使用数组保存状态转化的结果：

```
int tran[14000][2];
```

当  $W=3$  时，有 12 种状态，如表 6-3 所示：

表 6-3 当宽度为 3 时的状态转化情况

序号	十进制		二进制	
	from	to	from	to
0	7	6	111	110
1	6	7	110	111
2	7	3	111	011
3	7	0	111	000
4	6	1	110	001
5	5	2	101	010
6	4	3	100	011
7	3	7	011	111
8	3	4	011	100
9	2	5	010	101
10	1	6	001	110
11	0	7	000	111

(2) 从第 1 层向第  $n$  层, 利用动态规划算法递推结果

变量  $from$  和  $to$  的最大值, 就是宽为  $W$  的方格中都填充 1, 即  $(1 \ll W) - 1$ , 或者  $2^W - 1$ 。

初始时, 假设第 0 层都填充 1, 只有一种情况。

使用数组  $b$  表示每一层的递推关系, 递推公式如下。

令  $from[j] = tran[j][0]$ ,  $to[j] = tran[j][1]$ ,  $0 \leq j < nTran$

则 
$$b[i+1][to[j]] = \sum_{0 \leq j < nTran} b[i][from[j]], \quad 0 \leq j < nTran, \quad 0 \leq i < H。$$

当高  $H$  为 6 层, 宽  $W$  为 3 时, 递推的结果如图 6-3 所示。

		0	1	2	3	4	5	6	7
层编号 $H$	0	0	0	0	0	0	0	0	1
	1	1	0	0	1	0	0	1	0
	2	0	1	0	0	1	0	0	3
	3	3	0	0	4	0	0	4	0
	4	0	4	0	0	4	0	0	11
	5	11	0	0	15	0	0	15	0
	6	0	15	0	0	15	0	0	41

图 6-3 高  $H$  为 6 层, 宽  $W$  为 3 时, 递推的结果

最优值在  $b[H][(1 \ll W) - 1]$  单元。

当高  $H$  为 10 层, 宽  $W$  为 11 时, 最优值是:

3852472573499

其十六进制值是: 380F9425E3B, 需要 6 个字节存放, `int` 型数据就不行了。使用 8 字节的整形数就可以, 在 VC++ 中是 `_int64` (北京大学 JudgeOnline 是支持的, 参考代码见光盘 `zju1100-int64.c`), 其他编译器 (如 MinGW Developer Studio) 中是 `long long` (浙江大学 JudgeOnline 是支持的, 参考代码见光盘 `zju1100.cpp`)。最简单的办法是使用 `double` 型数据 (结果在有效位数之内)。

当高  $H$  为 11 层, 宽  $W$  为 11 时, 单元个数是奇数, 砖块填充不满, 结果为 0。

当宽  $W$  为 11 时,  $2^W - 1$  是 2047, 因此数组  $b$  的定义为:

```
double b[13][2100];
```

### 【程序代码】

```
程序名称:      zju1100.c
题    目:      Mondriaan's Dream
提交语言:      C++
运行时间:      00:00.00
运行内存:      500K
```

```
#include <stdio.h>
#include <string.h>
```

```
double b[13][2100];
int tran[14000][2];
```

```
//存放各层的递推结果
//保存状态转化的结果
```

```

int H, W, maxMove, nTran;
//深度优先搜索算法, 计算第一层的填充状态
//形参: 第 n 个方格, from 表示在这一层的编码, to 表示下一层的编码
void dfs (int n, int from, int to)
{
    if (n > W) return;
    //整个宽度 W 填充完毕
    if (n == W)
    {
        //保存当前的状态转化
        tran[nTran][0] = from;
        tran[nTran ++][1] = to;
        return;
    }
    dfs (n+2, (from<<2)+3, (to<<2)+3);    //水平放置砖块
    dfs (n+1, (from<<1)+1, to<<1);        //竖直放置砖块
    dfs (n+1, from<<1, (to<<1)+1);        //不放置砖块
}

//从第 1 层向第 n 层, 利用动态规划算法递推
void dp ()
{
    memset (b, 0x00, sizeof (b));
    //第 0 层都填充 1, 只有一种情况
    b[0][(1<<W) - 1] = 1;
    //实现递推公式
    int i, j;
    for (i = 0; i < H; i ++)
        for (j = 0; j < nTran; j ++)
            b[i+1][tran[j][1]] += b[i][tran[j][0]];
}

int main ()
{
    while (scanf ("%d%d", &H, &W) && H)
    {
        //确保 W < H
        int temp;
        //确保高度 H 大于宽度 W, 因为计算状态转移要费时一些
        if (H < W)
        {
            temp = H; H = W; W = temp;
        }
        nTran = 0;
        //调用深度优先搜索算法
    }
}

```

```

    dfs (0, 0, 0);
    //调用 DP 算法
    dp ();
    //结果在 b[H][2W- 1]单元中
    printf("%.0f\n", b[H][(1<W) - 1]);
}
return 0;
}

```

## ZJU1102-Phylogenetic Trees Inherited<sup>[1、2、3]</sup>

Time Limit: 10 Seconds

Memory Limit: 32768 KB

**Special Judge**

Among other things, Computational Molecular Biology deals with processing genetic sequences. Considering the evolutionary relationship of two sequences, we can say that they are closely related if they do not differ very much. We might represent the relationship by a tree, putting sequences from ancestors above sequences from their descendants. Such trees are called phylogenetic trees.

Whereas one task of phylogenetics is to infer a tree from given sequences, we'll simplify things a bit and provide a tree structure-this will be a complete binary tree. You'll be given the  $n$  leaves of the tree. Sure you know,  $n$  is always a power of 2. Each leaf is a sequence of amino acids (designated by the one-character-codes you can see in the figure). All sequences will be of equal length  $l$ . Your task is to derive the sequence of a common ancestor with minimal costs.

Amino Acid		
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic Acid	Asp	D
Cysteine	Cys	C
Glutamine	Gln	Q
Glutamic Acid	Glu	E
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I

Amino Acid		
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1102>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2414>

[3] <http://acm.uva.es/p/v101/10185.html>

The costs are determined as follows: every inner node of the tree is marked with a sequence of length  $l$ , the cost of an edge of the tree is the number of positions at which the two sequences at the ends of the edge differ, the total cost is the sum of the costs at all edges. The sequence of a common ancestor of all sequences is then found at the root of the tree. An optimal common ancestor is a common ancestor with minimal total costs.

### Input Specification

The input file contains several test cases. Each test case starts with two integers  $n$  and  $l$ , denoting the number of sequences at the leaves and their length, respectively. Input is terminated by  $n=l=0$ . Otherwise,  $1 \leq n \leq 1024$  and  $1 \leq l \leq 1000$ . Then follow  $n$  words of length  $l$  over the amino acid alphabet. They represent the leaves of a complete binary tree, from left to right.

### Output Specification

For each test case, output a line containing some optimal common ancestor and the minimal total costs.

### Sample Input

4 3	4 3	2 1	1 1
AAG	AAG	W	Q
AAA	GGA	W	
GGA	AAA		0 0
AGA	AGA	2 1	
		W	
4 3	4 1	Y	
AAG	A		
AGA	R		
AAA	A		
GGA	R		

### Sample Output

```
AGA 3
AGA 4
AGA 4
R 2
W 0
Y 1
Q 0
```

### Problem Source

University of Ulm Local Contest 2000

#### 【题目大意】

计算分子生物学（CMB）正在研究遗传基因链的数据处理。对于两种链之间的进化关系，

如果它们相差不大，我们就说它们有亲缘关系。我们可以用树形像地描述这种关系。祖先在上面，他们的后裔依次排列。这样的树叫做进化树。

种系遗传学的一个任务是从给出的生物链里推断出进化树。我们将把问题简化一些，给出一个树形结构和树的  $n$  片叶子，它是一棵完全二叉树。当然你得明白  $n$  总是 2 的指数。每片叶子都是含氨基酸的一个序列（如下面所示的单字符编码），所有序列的长度相等 ( $l$ )。你的任务是找出拥有共同祖先的序列，且花费最少。

计算花费的方法：每一个树的内结点都标记为长度  $l$  的一个序列。树每边的花费是指，该边两端的两个结点序列中，对应位置氨基酸不相同的个数。总耗费就是每边的花费总和。树的根结点就是所有序列公共的祖先。但最佳的公共祖先是总花费最小的公共祖先。

### 输入格式

输入有多组测试数据。对每组测试数据，第一行是两个整数  $n$  和  $l$ ，分别表示树叶的序列数及序列的长度。输入  $n=l=0$  结束。另外， $1 \leq n \leq 1024$ ， $1 \leq l \leq 1000$ 。接下来是含有含氨基酸字母的  $n$  行长度为  $l$  的字符串。从左边到右边，构成完全二叉树。

### 输出格式

对每组测试数据，输出最佳的公共祖先和最小总花费。

### 【算法分析】

本题的解答选自标程<sup>[1]</sup>，在该网页上还有其他的实现方法和裁判的算法分析，见光盘文件：Phylogenetic Trees Inherited.doc。

输入数据只给出了完全二叉树的所有叶子结点，没有给出内部结点。因此我们需要构造内部结点并计算边权。根据裁判的算法分析，采用动态规划的方法，能够高效地解决这一问题。自底向上构造  $n-1$  层的内结点，并计算边权；再构造  $n-2$  层的内结点，一直到第 1 层结点（也就是公共祖先）。这样构造出来的完全二叉树就是总边权最小的完全二叉树。构造结点时，如果两个孩子结点对应位置的字符（即氨基酸）相同，父结点对应位置就取该字符，否则取两个字符的并集（这样就导致了多种答案，所以是 Special Judge）。

#### （2）样例分析

以第一组输入数据为例：

```
4 3
AAG
AAA
GGA
AGA
```

其初始和最终的完全二叉树如图 6-4 所示。

构造结点 5 时，其子结点的序列中，最后一个字符不同，作为一个并集 (G, A) 放在那里，也就是取 G 或者 A 都可以。因为只有一个字符不同，所以花费为 1。构造结点 6 也是一样的，花费也是 1。构造结点 7 时，对第一个字符，左孩子是 A，右孩子是并集 (G, A)，就取交集 A，没有花费；第三个字符方法相同；而第二个字符是不同的，取并集 (G, A)，花费是 1。总花费是 3。因此答案：AGA 3 或者 AAA 3 都是正确答案。

---

[1] <http://www.informatik.uni-ulm.de/acm/Locals/2000/html/judge.html>

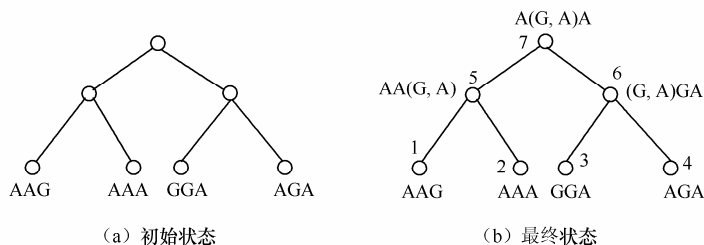


图 6-4 样例 1 数据分析

### (3) 数据结构

数组 `seq` 保存叶子结点序列为

```
char seq[1024][1024];
```

叶子结点的个数和序列的长度为

```
int n, len;
```

每次只计算序列中一个位置的字符，使用一维数组 `heap` 保存完全二叉树中所有该位置的字符（含并集）为

```
int heap[2048];
```

将字符（氨基酸）映射为整数：

31	30	29	28	27	26	25	24	.....	3	2	1	0
0	0	0	0	0	Z	Y	X		D	C	B	A

如果是 A，则值为 1；如果是 AC，则值为 5。这就将字符的交集/并集运算，转换为整数的按位与/或运算。

一般情况下，是用  $n$  维方阵表示完全二叉树，将完全二叉树存储在  $n$  维方阵的左下角。这种方法由于是二维结构，编码方便，可读性好。这里是用一维矩阵表示的，如图 6-5 所示。

$n$  个叶子结点，放在从下标  $n$  开始的  $n$  个单元中。

### (4) 动态规划算法的实现

建立递推关系：

对任意一个内结点  $k$ ，由于是完全二叉树，其左树叉的编号是  $2k$ ，右树叉的编号是  $2k+1$ 。父结点  $k$  的相应位置字符的生成，是根据其子结点的字符确定的，分为三种情况：

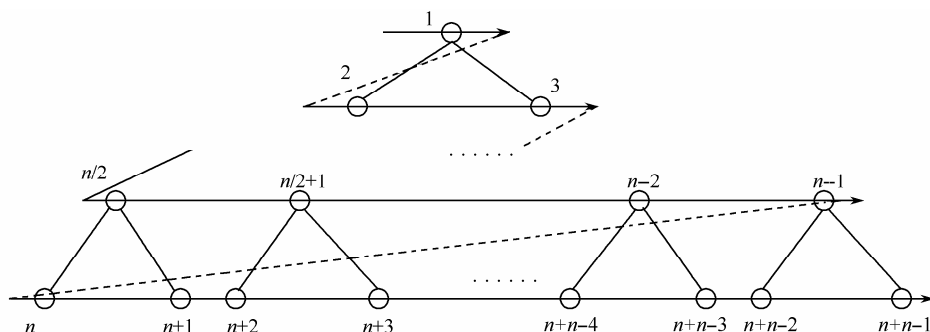


图 6-5 完全二叉树的一维数组表示法

① 两个子结点的字符是相同的，花费为 0，如图 6-6 所示。



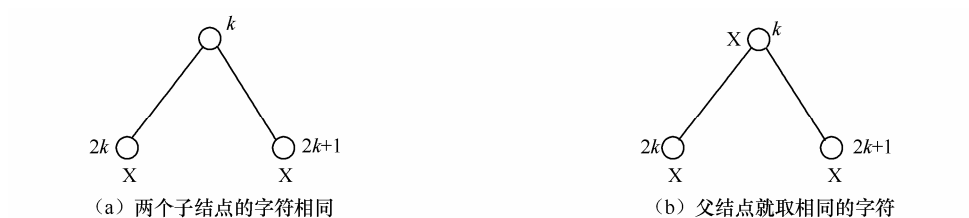


图 6-6 两个子结点的字符相同

② 两个子结点的字符是不相同的，花费为 1，如图 6-7 所示。



图 6-7 两个子结点的字符不相同

③ 一个子结点或者两个子结点的字符是并集，如图 6-8 所示。

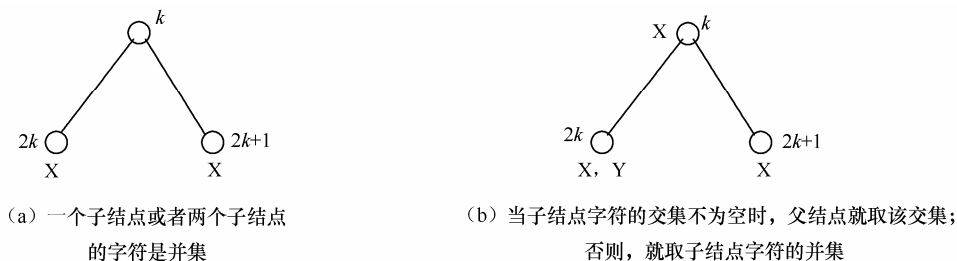


图 6-8 两个子结点的字符是并集

一个子结点或者两个子结点的字符是并集，先计算它们的交集。若交集不为空，则父结点就是该交集，花费为 0；若交集为空，则父结点就两个子结点的并集，花费为 1。

从叶子结点开始向上递推，最优解在根结点中，即 `heap[1]` 单元。

其实，第③种情况包含了前面两种情况，这样描述，读者应该容易理解。

#### (5) 并集字符的输出

并集的每个字符都是答案，显然输出并集中序号最大或者最小的字符是最容易的：

##### ● 输出序号最小的字符

将 `heap[1]` 和 1 做位与运算，如果为 0 则将 `heap[1]` 右移，直到与运算不为 0 为止，右移的次数就是字符的序号。

```
char ch = 'A';
while (!(heap[1]&1)) {
    heap[1]>>=1;
    ++ch;
}
printf("%c", ch);
```

● 输出序号最大的字符

将 heap[1] 右移, 直到 heap[1]=0 为止, 右移的次数就是字符的序号。见下面代码。

【程序代码】

---

程序名称:	zju1102.c
题    目:	Phylogenetic Trees Inherited
提交语言:	C
运行时间:	10ms
运行内存:	1184KB

---

```
#include <stdio.h>

char seq[1024][1024];          //保存叶子结点序列

int main ()
{
    int heap[2048];             //保存完全二叉树中所有某个位置的字符(含并集)
    int n, len;                 //叶子结点的个数和序列的长度
    while (scanf("%d %d", &n, &len) && (n || len))
    {
        int i, k;
        for (k = 0 ; k < n ; k++) //读取叶子结点序列
            scanf("%s", seq[k]);
        int cost = 0;            //总花费
        //对所有结点, 计算每一位字符
        for (i = 0 ; i < len ; i++)
        {
            //将字符映射为整数, 构造完全二叉树的叶子结点
            for (k = 0 ; k < n ; k++)
                heap[n+k] = 1<<(seq[k][i] - 'A');
            //动态规划, 从叶子结点开始向上递推
            for (k = n-1 ; k > 0 ; k--)
            {
                //计算交集
                if (!(heap[k] = heap[k+k] & heap[k+k+1])) { //若交集为空
                    ++cost;                                //花费增加 1
                    //父结点就是两个孩子结点的并集
                    heap[k] = heap[k+k] | heap[k+k+1];
                }
            }
            //并集字符的输出, 这里是输出序号最大的字符
            char ch = 'A';
            while (heap[1]>=1) ++ch;
            printf("%c", ch);
        }

        printf(" %d\n", cost);          //输出总花费
    }
}
```

```

    }
    return 0;
}

```

## ZJU1107-FatMouse and Cheese<sup>[1]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768 KB

---

FatMouse has stored some cheese in a city. The city can be considered as a square grid of dimension  $n$ : each grid location is labelled  $(p, q)$  where  $0 \leq p < n$  and  $0 \leq q < n$ . At each grid location Fatmouse has hid between 0 and 100 blocks of cheese in a hole. Now he's going to enjoy his favorite food.

FatMouse begins by standing at location  $(0,0)$ . He eats up the cheese where he stands and then runs either horizontally or vertically to another location. The problem is that there is a super Cat named Top Killer sitting near his hole, so each time he can run at most  $k$  locations to get into the hole before being caught by Top Killer. What is worse — after eating up the cheese at one location, FatMouse gets fatter. So in order to gain enough energy for his next run, he has to run to a location which have more blocks of cheese than those that were at the current hole.

Given  $n$ ,  $k$ , and the number of blocks of cheese at each grid location, compute the maximum amount of cheese FatMouse can eat before being unable to move.

### Input Specification

There are several test cases. Each test case consists of

a line containing two integers between 1 and 100:  $n$  and  $k$

$n$  lines, each with  $n$  numbers: the first line contains the number of blocks of cheese at locations  $(0,0)$   $(0,1)$   $\dots$   $(0, n-1)$ ; the next line contains the number of blocks of cheese at locations  $(1,0)$ ,  $(1,1)$ ,  $\dots$ ,  $(1, n-1)$ , and so on.

The input ends with a pair of  $-1$ 's.

### Output Specification

For each test case output in a line the single integer giving the number of blocks of cheese collected.

### Sample Input

```

3 1
1 2 5
10 11 6
12 12 7

```

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1107>

## Output for Sample Input

37

## Problem Source

Zhejiang University Training Contest 2001

### 【题目大意】

FatMouse 在城市里储藏了一些奶酪。可以认为城市是一个边长为  $n$  的正方形网格：每个格子的位置标号是  $(p, q)$ ,  $0 \leq p < n$ ,  $0 \leq q < n$ 。在每个格子有一个洞，FatMouse 储藏着  $0 \sim 100$  块奶酪。现在它要享受美餐了。

FatMouse 从位置  $(0, 0)$  开始吃。吃完所到之处的奶酪之后，它将沿水平或垂直方向继续前进。问题是有只叫做 Top Killer 的猫守在它的洞穴附近。为了避免被 Top Killer 抓到，每次它最多只能跑  $k$  个网格。更糟糕的是：每吃完一处奶酪之后，FatMouse 就变得胖一些。为了得到充足的能量到达下一站，他必须去一个比现在洞穴中奶酪更多的位置。

给出  $n, k$ ，及每个网格中奶酪的块数。计算 FatMouse 在变肥得不能动之前，它最多能吃多少块奶酪。

### 输入格式

有多组测试数据。每组测试数据包括：

第一行是  $1 \sim 100$  之间的两个整数： $n$  和  $k$

$n$  行，每行  $n$  个数：第一行是位置  $(0, 0) (0, 1) \cdots (0, n-1)$  奶酪的块数；下一行是位置  $(1, 0) (1, 1), \cdots, (1, n-1)$  奶酪的块数，等等。

输入以一对一结束。

### 输出格式

对每组测试数据，输出一行，是 FatMouse 吃的奶酪块数。

### 【算法分析】

为了说明方便，将题目从  $(0, 0)$  开始的坐标改成从  $(1, 1)$  开始的坐标。题目中给出一个  $n \times n$  的网格，FatMouse 在网格  $(1, 1)$ 。它每次在同一个方向上最多可以移动  $k$  步，而且每次所到网格上的数字都要比上一次网格上的数字大。累计 FatMouse 所经过的网格上的数字输出即可。

#### (1) 数据结构

网格的大小和每次最多移动的步数：

```
int n, k;
```

网格矩阵，其值是该位置保存的奶酪数量：

```
int grid[105][105];
```

#### (2) 记忆式搜索

记忆式搜索算法也称为备忘录方法，是动态规划算法的变形，它使用二维数组保存已经解决的子问题的答案，在下次需要解决此子问题时，只要简单地查看该子问题的解答，而不必重新计算。使用数组 mem 保存已经计算的结果：

```
int mem[105][105];
```

实现记忆式搜索的函数是：

```
int memSearch(int r, int c)
```

其中形参(int r, int c)是当前的搜索位置。

从 FatMouse 当前的位置开始，每次向四周移动一步，直到移动  $k$  步为止。在每次移动时，判断下一步是不是在网格内，下一个位置奶酪的数量是不是比当前多，并且到下一个位置所获得的奶酪总数是不是比当前位置多。

## 【程序代码】

程序名称：	zju1107.c
题    目：	FatMouse and Cheese
提交语言：	C
运行时间：	30ms
运行内存：	244KB

```
#include <stdio.h>
#include <string.h>
```

```
int n;                //网格的大小
int k;                //每次最多移动的步数
int grid[105][105];   //保存奶酪数量的网格矩阵
int mem[105][105];    //用于记忆式搜索，保存中间搜索结果
```

//实现记忆式搜索，形参是当前的搜索位置

```
int memSearch(int r, int c)
{
    int r1, c1;        //FatMouse 将要到达的下一个位置
    int i;
    int max = 0;        //FatMouse 吃掉的奶酪
    //如果不是-1，表示该网格已经搜索过，直接读取结果
    if(mem[r][c] != -1) return mem[r][c];
    //每次让 FatMouse 前进 1 步，直到 k 步
    for(i = 1; i <= k; i++){
        //向上前进 1 步
        r1 = r - i;
        //如果下一步是有效的，且下一步奶酪的数量比当前多
        if(r1 >= 1 && r1 <= n && grid[r][c] < grid[r1][c]){
            //移动到下一步，继续搜索
            mem[r1][c] = memSearch(r1, c);
            //移动到下一步后，奶酪总数是增长的
            if(mem[r1][c] > max) max = mem[r1][c];
        }
        //向下前进 1 步
        r1 = r + i;
        if(r1 <= n && r1 >= 1 && grid[r][c] < grid[r1][c]){
```

```

        mem[r1][c] = memSearch(r1, c);
        if(mem[r1][c] > max) max = mem[r1][c];
    }
    //向左前进 1 步
    c1 = c - i;
    if(c1 >= 1 && c1 <= n && grid[r][c] < grid[r][c1]){
        mem[r][c1] = memSearch(r, c1);
        if(mem[r][c1] > max) max = mem[r][c1];
    }
    //向右前进 1 步
    c1 = c + i;
    if(c1 <= n && c1 >= 1 && grid[r][c] < grid[r][c1]){
        mem[r][c1] = memSearch(r, c1);
        if(mem[r][c1] > max) max = mem[r][c1];
    }
}
return max + grid[r][c]; //吃掉当前格子中的奶酪
}

int main()
{
    int i, j;
    while(scanf("%d%d", &n, &k)){
        if(n == -1 && k == -1) break;
        //读取数据, 建立 grid 矩阵
        for(i = 1; i <= n; i++)
            for(j = 1; j <= n; j++)
                scanf("%d", &grid[i][j]);
        //用于搜索的 mem 数组初始化, -1 表示该网格没有搜索过
        memset(mem, -1, sizeof(mem));
        printf("%d\n", memSearch(1, 1));
    }
    return 0;
}

```

## ZJU1108-FatMouse's Speed<sup>[1]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

**Special Judge**

---

FatMouse believes that the fatter a mouse is, the faster it runs. To disprove this, you want to take the data on a collection of mice and put as large a subset of this data as possible into a sequence

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1108>

so that the weights are increasing, but the speeds are decreasing.

## Input Specification

Input contains data for a bunch of mice, one mouse per line, terminated by end of file.

The data for a particular mouse will consist of a pair of integers: the first representing its size in grams and the second representing its speed in centimeters per second. Both integers are between 1 and 10000. The data in each test case will contain information for at most 1000 mice.

Two mice may have the same weight, the same speed, or even the same weight and speed.

## Output Specification

Your program should output a sequence of lines of data; the first line should contain a number  $n$ ; the remaining  $n$  lines should each contain a single positive integer (each one representing a mouse). If these  $n$  integers are  $m[1], m[2], \dots, m[n]$  then it must be the case that

$$W[m[1]] < W[m[2]] < \dots < W[m[n]]$$

and

$$S[m[1]] > S[m[2]] > \dots > S[m[n]]$$

In order for the answer to be correct,  $n$  should be as large as possible.

All inequalities are strict: weights must be strictly increasing, and speeds must be strictly decreasing. There may be many correct outputs for a given input, your program only needs to find one.

## Sample Input

```
6008 1300
6000 2100
500 2000
1000 4000
1100 3000
6000 2000
8000 1400
6000 1200
2000 1900
```

## Output for Sample Input

```
4
4
5
9
7
```

## Problem Source

Zhejiang University Training Contest 2001

### 【题目大意】

FatMouse 相信：长得越胖的老鼠跑得越快。为了证明这是不对的，你需要搜集老鼠的数据。然后，在这些数据中选取一个尽可能大的子集，从而发现体重不断增长时，速度却不断下降。

#### 输入格式

输入一群老鼠的资料，每只老鼠占一行。到达文件结尾时，输入结束。

每只老鼠的数据是一对整数：第一个表示它的体重（克），第二个表示它的速度（厘米/秒），两个整数的范围是 1~10000。每组测试数据最多包含 1000 只老鼠的信息。

任何两只老鼠有可能体重相同，或速度相同，甚至体重和速度都完全相同。

#### 输出格式

你的程序应该输出一系列数据：第一行是数字  $n$ ；接着有  $n$  行，每行是一个正整数（每个代表一只老鼠）。如果这  $n$  个整数是  $m[1], m[2], \dots, m[n]$ ，则必须满足：

$$W[m[1]] < W[m[2]] < \dots < W[m[n]]$$

和

$$S[m[1]] > S[m[2]] > \dots > S[m[n]]$$

为了确保答案是正确的， $n$  应该尽可能的大。

所有的不等式都严格遵守约束：重量一定严格的逐渐增加，速度一定严格的逐渐减少。可能有多种正确输出，你的程序只需输出其中一个。

### 【算法分析】

本题要求对一群老鼠的资料进行整理，在所有数据中找出一个最大的子集，确保重量一定严格的逐渐增加，速度一定严格的逐渐减少。

因为有两个序列方向，直接应用最长单调递增子序列算法是不行的。对数据进行预处理，首先按重量升序，在重量相同时，按速度降序，然后对速度序列应用最长单调递增子序列算法。

#### （1）样例分析

在按重量升序、重量相同时按速度降序排列后的结果如表 6-4 所示：

表 6-4 样例数据排序之后的结果

序号	重量 (weight)	速度 (speed)	原序列编号 (id)
1	500	2000	3
2	1000	4000	4
3	1100	3000	5
4	2000	1900	9
5	6000	2100	2
6	6000	2000	6
7	6000	1200	8
8	6008	1300	1
9	8000	1400	7



可以看出,满足要求的序列最大长度是4。在序号5、6和7的位置,三个重量都是6000,只能取一个速度。在长度为4的情况下,显然答案有多组:

- ① 4, 5, 2, 1
- ② 4, 5, 2, 7
- ③ 4, 5, 6, 1
- ④ 4, 5, 6, 7
- ⑤ .....

(2) 数据结构

采用结构体表示老鼠的资料:

```
struct mouse {
    int weight, speed, id;           //老鼠的重量、速度和编号
} mice[1001];
```

老鼠的实际数量用变量 *n* 表示。

(3) 排序算法的实现

采用 C 语言的 `qsort()` 排序:

```
qsort(mice, n, sizeof(mouse), cmp);
```

其中 `cmp` 是排序因子。在排序因子函数中,首先按老鼠的重量升序,在重量相同时,按速度降序。

(4) 最长单调递增子序列算法的实现

有关最长单调递增子序列算法,请参考相关的算法分析书籍。

实现时需要两个辅助数组:

- ① 存储构造到第 *i* ( $1 \leq i < n$ ) 个老鼠时,序列的最大长度

```
int count[1001] = {0};
```

对样例数据,数组 `count` 的值如下表 6-5 所示:

表 6-5 样例数据数组 `count` 的值

序	1	2	3	4	5	6	7	8	9
值	1	1	2	3	3	3	4	4	4

由于相同的数字很多,就知道为什么答案不是唯一的了。

- ② 存储构造到第 *i* ( $1 \leq i < n$ ) 个老鼠时,序列的前驱

```
int path[1001] = {0};
```

对样例数据,数组 `path` 的值如下表 6-6 所示:

表 6-6 样例数据数组 `path` 的值

序	1	2	3	4	5	6	7	8	9
前驱	0	0	2	3	3	3	4	4	4
原序号	3	4	5	9	2	6	8	1	7

(5) 输出子序列的序号

首先查找最大的序列长度,也就是在数组 `count` 中查找最大值 `max`,并记录该最大值所在的位置 `pos`。

有了位置 `pos`,在数组 `path` 中递归输出就可以了。如果当前位置是 `pos`,则前一个位置是

path[pos]。采用递归算法，刚好确保从前往后的顺序输出。

### 【程序代码】

---

程序名称:	zju1108.cpp
题    目:	FatMouse's Speed
提交语言:	C++
运行时间:	0ms
运行内存:	188KB

---

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

struct mouse {
    int weight, speed, id;    //老鼠的重量、速度和编号
} mice[1001];                //存储所有老鼠的资料

//排序因子
int cmp(const void * a, const void * b)
{
    mouse* ta = (mouse*) a;
    mouse* tb = (mouse*) b;
    if(ta -> weight == tb -> weight)        //重量相同时
        return tb -> speed - ta -> speed;    //按速度降序
    return ta -> weight - tb -> weight;        //按重量升序
}

void output(int path[], int pos)
{
    if(pos == 0) return;
    output(path, path[pos]);
    printf("%d\n", mice[pos].id);
}

int main()
{
    //读取并构造所有老鼠的资料
    int n = 1;                                //老鼠的数量
    while(scanf("%d%d", &mice[n].weight, &mice[n].speed) != EOF)
    {
        mice[n].id = n;                        //构造编号
        n++;
    }
    qsort(mice, n, sizeof(mouse), cmp);        //排序

    //最长单调递增子序列算法的实现，下面是针对排序后的数组
```

```

//存储构造到第 i (1≤i<n) 个老鼠时, 序列的最大长度
int count[1001] = {0};
//存储构造到第 i (1≤i<n) 个老鼠时, 序列的前驱
int path[1001] = {0};
count[1] = 1;
for(int i = 2; i < n; i++)
{
    for(int j = 1; j < i; j++)
        if(mice[i].weight > mice[j].weight && mice[i].speed < mice[j].speed)
            if(count[i] < count[j])                //得到了更长的序列
            {
                count[i] = count[j];
                path[i] = j;                        //记录前驱
            }
    count[i]++;
}
//查找最大的序列长度
int max = 0;                                       //最大长度
int pos;                                          //最大长度元素所在位置
for(int i = 1; i < n; i++)
    if(count[i] > max)
    {
        max = count[i];
        pos = i;
    }
printf("%d\n", max);                             //输出最大长度
output(path, pos);                               //输出子序列的序号
return 0;
}

```

## ZJU1132-Railroad<sup>[1、2、3]</sup>

---

Time Limit: 10 Seconds      Memory Limit: 32768 KB

---

It is Friday evening and Jill hates two things which are common to all trains:

1. They are always late.
2. The posted schedule is always wrong.

Nevertheless, tomorrow in the early morning hours Jill will have to travel from Tuttlingen to Freiburg in order to get to the Regional Programming Contest. Since she is afraid of arriving too late and being excluded from the contest, she is looking for the train which gets her to Freiburg as early

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1132>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1394>

[3] <http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2200>

as possible.

However, she dislikes getting to the station too early, so if there are several schedules with the same arrival time, she will choose the one with the latest departure time.

Jill asks you to help her with her problem, so that she can sleep a bit longer tomorrow. You are given a set of railroad schedules from which you have to compute the fastest connection among those with the earliest arrival time for going from one location to another. One good thing: Jill is very experienced in switching trains: she can do this instantaneously, i.e., in zero time!!!

## Input

The input file contains several scenarios. Each of them consists of three parts.

Part one lists the names of all cities connected by the railroads. It starts with a line containing an integer  $C$  ( $1 \leq C \leq 100$ ) followed by  $C$  lines containing city names. These names consist of letters.

Part two describes all the trains running during the day. It starts with a number  $T \leq 1000$  followed by  $T$  train descriptions. Each train description consists of one line with a number  $t_i \leq 100$  and  $t_i$  more lines with a time and a city name, meaning that passengers can get on or off the train at that time at that city. The times are given in the 24-hour format hhmm.

Part three consists of three lines: Line one contains the earliest possible starting time for the journey, line two the name of the city where she starts, and line three the destination city. The two cities are always different.

The end of the input file is marked by a line containing only a zero (instead of  $C$ ). Do not process this line.

## Output

For each scenario print the line “Scenario #n” where n is the number of the scenario starting at 1.

If a connection exists then print the two lines containing zero padded timestamps and locations as shown in the sample output. Use blanks to achieve the indentation. If no connection exists on the same day (i.e., arrival before midnight), then print a line containing "No connection".

After each scenario print a blank line.

## Sample Input

3	2
Tuttlingen	Ulm
Constance	Vancouver
Freiburg	1
3	2
2	0100 Ulm
0949 Tuttlingen	2300 Vancouver
1006 Constance	0800
2	Ulm
1325 Tuttlingen	Vancouver
1550 Freiburg	0

```
2
1205 Constance
1411 Freiburg
0800
Tuttlingen
Freiburg
```

## Sample Output

```
Scenario #1
Departure 0949 Tuttlingen
Arrival 1411 Freiburg
Scenario #2
No connection
```

## Problem Source

Mid-Central European Regional Contest 2000

### 【题目大意】

又到星期五晚上了，令 Jill 烦心的两件事又来了——几乎所有的火车都会发生：

- ① 火车总是晚点。
- ② 火车时刻表总是错的。

然而，明天清晨，Jill 要从 Tuttlingen 乘车到 Freiburg，参加地区程序设计大赛。她害怕因为迟到而被取消参赛资格，她正在找能尽早到达 Freiburg 的火车。

但是，她不喜欢太早赶到火车站。因此，如果有几趟火车到达时间相同的话，她会选择最晚出发的那辆火车。

Jill 希望你能帮助她解决这个问题。这样的话，她明天就能多睡一会儿了。根据给出的火车时刻表，你需要计算能够最快最早到达另一个地方的旅行方式。一条好消息：在换乘火车方面，Jill 富有经验：她能够瞬间换车，时间为零!!!

### 输入格式

输入有多组测试例。每组测试例，分为三个部分：

第一部分是由铁路连接的所有城市名。第一行是一个整数  $C(1 \leq C \leq 100)$ ，接下来的  $C$  行都是城市名，全部是字母。

第二部份是全天运行的所有火车。开头是整数  $T \leq 1000$ ，接下来就是  $T$  辆火车。每辆火车的第一行是数字  $t_i \leq 100$ ，然后是不少于  $t_i$  行的时间和城市名，意思是乘客能在该时间该城市上下车。所给的时间是 24 小时制。

第三部份有三行：第一行是旅行的最早可能出发时间，第二行是起点站的城市名，第三行是目的地城市名。这两个城市是不同的。

当一行是 0(代替  $C$ )时，输入结束，该行无需处理。

### 输出格式

对每组测试数据，输出 “Scenario #n”， $n$  是测试例编号，从 1 开始。

如果存在旅行路线，则输出两行，包括时间和地点（如输出样例所示）。使用空格对齐位置。如果当天无旅行路线（也就是，不能在午夜之前抵达），输出 “No connection”。

每组测试数据之后空一行。

【算法分析】

在起点城市，Jill 有很多火车可以选择。编程任务是计算，在 Jill 最早可能出发的时间之后，选择最晚出发的火车，而又要能够最快最早到达目的地的旅行方式。编程工作量显然不小。

(1) 数据结构

城市的数量用变量  $C$  表示，火车的数量用变量  $T$  表示。

使用数组 `name` 表示城市名称：

```
char name[100][100];
```

该数组的下标就是城市的编号。对于样例数据 1，数组 `name` 的值如表 6-7 所示：

表 6-7 样例数据 1 的城市编号

下标	0	1	2
城市	Tuttlingen	Constance	Freiburg

查找城市编号的任务，由函数实现：

```
int search(char *cityName)
```

形参 `cityName` 是待查的城市编号。在数组 `name` 中查找城市 `cityName`，找到了就直接返回数组下标，否则返回-1。

如果使用 C++ 的标准模板库函数 `map()`，这项任务就变得非常简单，如 ZJU1129-Erdos Numbers 中对作者姓名的编号。

使用数组 `train` 表示每趟火车的换乘站数：

```
int train[1000];
```

对于样例数据 1，数组 `train` 的值如表 6-8 所示：

表 6-8 样例数据 1 的每趟火车的换乘站数

火车编号	0	1	2
换乘站数	2	2	2

使用数组 `change` 表示每趟火车换乘关系的邻接矩阵：

```
struct {  
    int time, city; //换乘时间和换乘城市的编号  
} change[1000][100];
```

对于样例数据 1，数组 `change` 的值如表 6-9 所示：

表 6-9 样例数据 1 的火车换乘关系邻接矩阵

火车编号	换乘 0		换乘 1	
	时间	城市编号	时间	城市编号
0	949	0	1006	1
1	1325	0	1550	2
2	1205	1	1411	2

(2) 计算最佳旅行方式

本题有两个要求：①最早到达目的地；②要求在最早可能出发的时间之后，尽可能最晚乘车。这要分两步实现：

① 在最早可能出发的时间之后，最早到达目的地时间

由于起点有很多火车，最简单的办法就是使用 Floyd-Warshall 算法。关于该算法，可以参考清华大学出版社，严蔚敏编写的《数据结构》教材。下列网站上的资料，推荐阅读：

<http://baike.baidu.com/view/14495.htm>

[http://en.wikipedia.org/wiki/Floyd-Warshall\\_algorithm](http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm)

使用数组 `travel` 存放每个城市的最佳换乘时间：

```
int travel[100];
```

数组的初值是  $\infty$ ，城市 `start` 的换乘时间是 `departure`，即 Jill 旅行的最早可能出发时间。

对每一个城市  $k$  ( $k = 0 \sim C-1$ )，挑选换乘时间最快的火车，如图 6-9 所示：

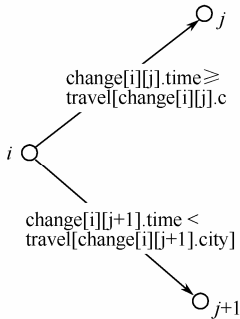


图 6-9 挑选换乘时间最快的火车

图中 `change[i][j].time` 表示火车  $i$  第  $j$  个换乘的时间，`travel[change[i][j].city]` 表示该换乘当前的最佳时间。算法表示，对每趟火车，搜索最佳的换乘站点。

对样例数据 1，数组 `travel` 的值如表 6-10 所示：

表 6-10 样例数据 1 的每个城市的最早换乘时间

城市编号	0	1	2
换乘时间	800	1006	1411

从表中看出，到达 Freiburg（城市编号是 2）的最早时间是 1411。

显然搜索结束时，就得到了 Jill 最早到达目标城市 `dist` 的时间。

② 在最早可能出发的时间之后，尽可能最晚乘车的时间

除 `dist` 单元外，数组 `travel` 初始化为  $-1$ 。对每一个城市  $k$  ( $k = 0 \sim C-1$ )，在满足上一步计算的最早到达时间（`travel[dist]`）的情况下，挑选换乘时间最晚的火车，其计算方法与第①步是相同的。

对样例数据 1，数组 `travel` 的值如表 6-11 所示：

表 6-11 样例数据 1 的每个城市的最晚换乘时间

城市编号	0	1	2
换乘时间	949	1205	1411

从表中看出，从 Tuttlingen（城市编号是 0）出发的最晚时间是 949；在城市 Constance（城

市编号是 1) 换乘的最晚时间是 1205。

### 【程序代码】

---

程序名称:	zju1132.c
题    目:	Railroad
提交语言:	C
运行时间:	120ms
运行内存:	956KB

---

```
#include <stdio.h>
#include <memory.h>

int C,T;                                     //城市的数量, 火车的数量
struct {
    int time, city;                          //换乘时间和换乘城市的编号
} change[1000][100];                        //火车换乘关系的邻接矩阵
char name[100][100];                        //城市名称
int start, dist;                             //起点城市和目标城市编号
int train[1000];                             //每趟火车的换乘站数
int travel[100];                             //每个城市的最佳换乘时间

//查找城市编号
int search (char *cityName)
{
    int i;
    for (i = 0; i<C; i++)
        if (!strcmp(name[i], cityName))
            return i;
    return -1;
}

//计算最佳旅行方式, 形参 departure 是城市 start 的换乘时间
void solve(int departure)
{
    int i,j,k;
    //在最早可能出发的时间之后, 计算最早到达目的地时间
    for (i = 0; i<C; i++)
        travel[i] = 1000000;
    travel[start] = departure;
    for (k = 0; k<C; k++)
        for (i = 0; i<T; i++)
            for (j = 0; j<train[i]-1; j++)
                if (change[i][j].time>=travel[change[i][j].city]
                    && travel[change[i][j+1].city]>change[i][j+1].time)
                    travel[change[i][j+1].city] = change[i][j+1].time;
```



```

//在最早可能出发的时间之后，尽可能最晚乘车的时间，且满足最早到达时间
for (i = 0; i<C; i++)
    if (i!=dist)
        travel[i] = -1;
for (k = 0; k<C; k++)
    for (i = 0; i<T; i++)
        for (j = 0; j<train[i]-1; j++)
            if (change[i][j+1].time<=travel[change[i][j+1].city]
                && travel[change[i][j].city]<change[i][j].time)
                travel[change[i][j].city] = change[i][j].time;
}

int main()
{
    int i,j;
    char str[1000];
    int departure; //最早可能出发的时间
    int iCase = 0; //测试例编号

    while (scanf("%d",&C) && C)
    {
        for (i = 0; i<C; i++) //读取城市名称
            scanf("%s", name[i]);
        scanf("%d",&T);
        //构造火车换乘关系的邻接矩阵
        for (i = 0; i<T; i++)
        {
            scanf("%d", &train[i]);
            for (j = 0; j<train[i]; j++)
            {
                scanf("%d%s", &change[i][j].time, str);
                change[i][j].city = search(str);
            }
        }
        scanf("%d",&departure);
        scanf("%s",str);
        start = search(str); //出发城市
        scanf("%s",str);
        dist = search(str); //到达城市
        //计算最佳旅行方式
        solve(departure);
        //输出结果
        printf("Scenario #%d\n",++iCase);
        if (travel[dist]<1000000)

```

```

    {
        printf("Departure %.4d %s\n", travel[start], name[start]);
        printf("Arrival    %.4d %s\n\n", travel[dist], name[dist]);
    }
    else
        printf("No connection\n\n");
}
return 0;
}

```

## ZJU1147-Formatting Text<sup>[1、2、3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

Writings e-mails is fun, but, unfortunately, they do not look very nice, mainly because not all lines have the same lengths. In this problem, your task is to write an e-mail formatting program which reformats a paragraph of an e-mail (e.g. by inserting spaces) so that, afterwards, all lines have the same length (even the last one of each paragraph).

The easiest way to perform this task would be to insert more spaces between the words in lines which are too short. But this is not the best way. Consider the following example:

```

*****
This is the example you are
actually considering.

```

Let us assume that we want to get lines as long as the row of stars. Then, by simply inserting spaces, we would get

```

*****
This is the example you are
actually      considering.

```

But this looks rather odd because of the big gap in the second line. By moving the word "are" from the first to the second line, we get a better result:

```

*****
This is the example you
are actually considering.

```

Of course, this has to be formalized. To do this, we assign a badness to each gap between words. The badness assigned to a gap of  $n$  spaces is  $(n - 1)^2$ . The goal of the program is to minimize the sum of all badnesses. For example, the badness of the first example is  $1 + 7^2 = 50$  whereas the badness of the second one is only  $1 + 1 + 1 + 4 + 1 + 4 = 12$ .

In the output, every line has to start and to end with a word. (I.e. there cannot be a gap at the

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1147>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1093>

[3] <http://acm.uva.es/p/v7/709.html>

beginning or the end of a line.) The only exception to this is the following:

If a line contains only one word this word shall be put at the beginning of the line, and a badness of 500 is assigned to this line if it is shorter than it should be. (Of course, in this case, the length of the line is simply the length of the word.)

## Input

The input contains a text consisting of several paragraphs. Each paragraph is preceded by a line containing a single integer  $n$ , the desired width of the paragraph ( $1 \leq n \leq 80$ ).

Paragraphs consist of one or more lines which contain one or more words each. Words consist of characters with ASCII codes between 33 and 126, inclusive, and are separated by spaces (possibly more than one). No word will be longer than the desired width of the paragraph. The total length of all words of one paragraph will not be more than 10000 characters.

Each paragraph is terminated by exactly one blank line. There is no limit on the number of paragraphs in the input file.

The input file will be terminated by a paragraph description starting with  $n=0$ . This paragraph should not be processed.

## Output

Output the same text, formatted in the way described above (processing each paragraph separately).

If there are several ways to format a paragraph with the same badness, use the following algorithm to choose which one to output: Let A and B be two solutions. Find the first gap which has not the same length in A and B. Do not output the solution in which this gap is bigger.

Output a blank line after each paragraph.

## Sample Input

```
28
This is the example you are
actually considering.

25
Writing e-mails is fun, and with this program,
they even look nice.

0
```

## Sample Output

```
This is the example you
are actually considering.

Writing e-mails is fun,
and with this program,
```

they even look nice.

## Problem Source

Mid-Central European Regional Contest 1999

### 【题目大意】

虽然写电子邮件很有趣，但由于每行单词的长度不同，邮件内容就显得不怎么美观。编程任务：写一个格式化电子邮件的程序，对电子邮件重新排版（例如，插入空格），以便使每行的长度都一样（即使是每个段落的最后一行）。

最简单的方法是在词句之间插入很多空格。但这不是最好的方法，如下例所示：

```
*****  
This is the example you are  
actually considering.
```

假设我们设计的句子长度和星号的那行一样。然后，通过插入空格，就会得到：

```
*****  
This is the example you are  
actually      considering.
```

但由于第二行有一个大的空格，这显得非常不自然。把单词“are”移到第二行，就会有一个较美观的格式：

```
*****  
This is the example you  
are actually considering.
```

当然，这样的格式比较好。为了做到这点，我们给单词之间的间隙引入一个不良度（badness）。一个有  $n$  个空格的间隙，不良度是  $(n-1)^2$ 。程序的目标是使所有间隙的不良度总和最小。例如，第一个例子的不良度是  $1+7^2=50$ ，而第二个例子是  $1+1+1+4+1+4=12$ 。

输出结果中，每行的开头和结尾都是一个单词（即在开头和结尾没有空格）。下面情况是例外：

如果一行中只有一个单词，那么这个单词直接顶行输出。如果该单词的长度小于行宽，其不良度为 500。（当然，这种情况下，这行的长度就是该单词的长度。）

### 输入格式

输入有多个段落。每个段落的开头是一个整数  $n$ ，表示段落宽度 ( $1 \leq n \leq 80$ )。

段落有一行或者多行，每行有一个或多个单词。单个单词由 ASCII 码 33~126 之间的字符表示，单词之间有空格。单词的长度小于段落宽度。每段所有单词的长度不超过 10000 个字符。

每个段落的末尾是一个空行。输入文件中，没有段落数的限制。

输入  $n=0$  时结束，不需处理。

### 输出格式

输出内容不变，格式如上所述。（每段单独处理）

如果不良度相同，有多种方法优化一个段落，使用下列算法来选择较好的一个：设 A 和 B 是两种解决方法。找出 A 和 B 中第一个长度不同的间隙，不要输出间隙长度较大的那种解决方法。

每个段落后输出一空行。

## 【算法分析】

单词全部读取以后，变成完整的一段（不管原来有几行），按照指定的宽度 **width** 重新排版。插入的空格数，要按不良度（**badness**）进行评估。在各种可行的方案中，要使用不良度最小的排版方案。因为排版时，一行只剩一个单词的情况，肯定是在最后一行，所以从最后一个单词开始排版。动态规划算法能够得到很好的计算效率。

### （1）数据结构与数据的读取

对一个段落（电子邮件）的所有单词，存放在数组中：

```
char word[1000][ 60];
```

每个单词的长度存放在数组中：

```
int len[1000];
```

排版宽度：

```
int width;
```

使用 **gets()** 每次读取一行，以空行控制段落的结束（字符串长度为 0）；对每行文本，用变量 **pos** 作为位置指针，使用 **sscanf()** 读取一个单词，将单词长度记到数组 **len** 中；同时 **pos** 跳到该单词的后面（累加该单词的长度），接着读取下一个单词。

### （2）动态规划算法的实现

由函数实现：

```
void DP(int num);
```

形参 **num** 是该段落的单词总数。

反向 DP，即倒着推。设从第  $i$  ( $0 \leq i < \text{num}$ ) 个单词起的 **line** [ $i$ ] 个单词作为第一行，从第 **line** [ $i$ ] +  $i$  个单词起的 **line** [**line** [ $i$ ] +  $i$ ] 个单词做为第二行，依此类推，如图 6-10 所示。

将相应排版的不良度存入数组 **badness** 中。

设变量  $j$  表示从第  $i$  个单词起的第  $j$  个单词 ( $1 \leq j \leq \text{num} - i$ )，并且第  $j$  个单词能够排版到该行中（控制该行字符和空格的长度），则有递推式：

```
line[i] = { j | min(badness(i, j)) }
```

```
( $0 \leq i < \text{num}$ ,  $1 \leq j \leq \text{num} - i$  且第  $j$  个单词能够排版到该行中)
```

```
badness(i, j) = badness[i + j] + score;
```

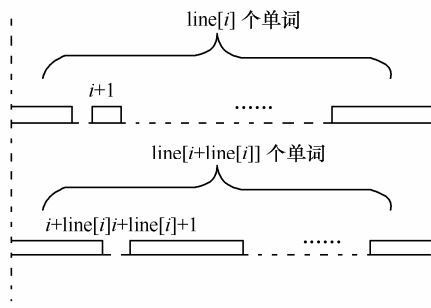


图 6-10 数组 **line** 的定义

式中 **score** 是当前行的 **badness**。

初值：**badness**[**num**] = 0。

对于样例数据 1，动态规划的结果如表 6-12 所示：

表 6-12 样例数据 1 的动态规划结果

单词	数组 line	数组 badness
This	5	12
is	4	39
the	4	76
example	3	134
you	2	490
are	3	5
actually	2	49
considering.	1	500

从表中看出，第 1 行排 5 个单词，第 2 行排 3 个单词，总的 badness 最小（12）。

（1）输出排版结果

根据数组 line 输出即可。设变量  $k$  表示每行的行首单词编号，则第一行， $k=0$ ；第二行， $k=\text{line}[0]$ ；第三行， $k+=\text{line}[1]$ ，依此类推。

空格的分配：设当前行单词字符的总长度是 total，则每个间隙中的最少空格数  $q$  是：

$$q = (\text{width} - \text{total}) / (j-1);$$

这样，还有  $r$  个空格未安排：

$$r = (\text{width} - \text{total}) \% (j-1);$$

需要  $r$  个间隙中每个间隙再多一个空格，放在该行的后面，就满足了题目要求的优化方法。

## 【程序代码】

---

程序名称:           zju1147.c  
 题    目:           Formatting Text  
 提交语言:           C  
 运行时间:           0ms  
 运行内存:           224KB

---

```
#include <stdio.h>
#include <string.h>

#define MAXN 1000
#define MAXM 60
const int INF = 999999999;           //∞
char word[MAXN][MAXM];              //一个段落（电子邮件），段落有多行
int width;                          //排版宽度
int len[MAXN];                      //每个单词的长度
//用于 DP 的数组，每个单词到行末的单词个数
int line[MAXN];

//动态规划算法的实现，形参 num 是单词总数
void DP(int num) {
    int i, j;
    //用于 DP 的数组，相应于数组 line 的每个单词的不良度（badness）
```

```

int badness[MAXN];
badness[num] = 0;
//从最后一个单词起的每个单词
for(i = num - 1; i >= 0; i--) {
    line[i] = 1;
    j = 1; //单词 i 本身
    int total = 0; //该行字符的长度
    int min = INF; //最小不良度
    //将单词 i 放在行首, 在一行内能够排版的单词数中,
    while(i+j-1 < num && total+len[i+j-1]+j-1 <=width) {
        total += len[i+j-1];
        //计算不良度, 初值为第 i+j 个单词的不良度
        int score = badness[i + j];
        if (j==1) score += 500; //只能放一个单词的不良度
        else {
            //每个间隙中的最少空格数
            int q = (width - total) / (j-1);
            //r 个间隙中, 每个间隙再多一个空格
            int r = (width - total) % (j-1);
            score += r*q*q + (j-1-r)*(q-1)*(q-1);
        }
        if(score <= min) { //最小不良度
            min = score;
            line[i] = j; //该单词到行末的单词个数
        }
        j++; //再放一个单词
    }
    badness[i] = min; //该单词的不良度
}

int main()
{
    int i, j;
    char ch[500]; //一行字符
    while(scanf("%d\n", &width) && width) {
        int word_num = 0; //单词数
        //读取一行, 且不是空行
        while(gets(ch) && strlen(ch)) {
            int pos = 0;
            while(ch[pos]) { //不是行末
                //读取一个单词
                sscanf(ch+pos, "%s", word[word_num]);
                //存放该单词的长度
                len[word_num] = strlen(word[word_num]);
                pos += strlen(word[word_num++]);
                while (ch[pos] && ch[pos]!=' ') pos++;
            }

```

```

    }
    DP(word_num);
    int k = 0; //每行第一个单词的序号
    while(k < word_num) {
        int total = 0; //该行中字符的长度
        for(i = k; i < k + line[k]; i++)
            total += len[i];
        //该行只有一个单词
        if(line[k] == 1) {
            printf("%s\n", word[k]);
            k++;
            continue;
        }
        //每个间隙中的最少空格数
        int q = (width - total) / (line[k] - 1);
        // r 个间隙中, 每个间隙再多一个空格, 放在该行的后面
        int r = (width - total) % (line[k] - 1);
        //输出该行的前 p[k] - 1 个单词
        for(i = k; i < k + line[k] - 1; i++) {
            printf("%s", word[i]);
            int space = q + (i >= k + line[k] - 1 - r);
            for (j=0; j<space; j++)
                putchar(' ');
        }
        //输出该行的最后一个单词
        printf("%s\n", word[line[k] + k - 1]);
        //下一行的行首单词的序号
        k += line[k];
    }
    printf("\n");
}
return 0;
}

```

## ZJU1149-Dividing<sup>[1、2、3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

Marsha and Bill own a collection of marbles. They want to split the collection among themselves so that both receive an equal share of the marbles. This would be easy if all the marbles had the same value, because then they could just split the collection in half. But unfortunately, some

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1149>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1014>

[3] <http://acm.uva.es/p/v7/711.html>



of the marbles are larger, or more beautiful than others. So, Marsha and Bill start by assigning a value, a natural number between one and six, to each marble. Now they want to divide the marbles so that each of them gets the same total value.

Unfortunately, they realize that it might be impossible to divide the marbles in this way (even if the total value of all marbles is even). For example, if there are one marble of value 1, one of value 3 and two of value 4, then they cannot be split into sets of equal value. So, they ask you to write a program that checks whether there is a fair partition of the marbles.

## Input

Each line in the input describes one collection of marbles to be divided. The lines consist of six non-negative integers  $n_1, n_2, \dots, n_6$ , where  $n_i$  is the number of marbles of value  $i$ . So, the example from above would be described by the input-line "1 0 1 2 0 0". The maximum total number of marbles will be 20000.

The last line of the input file will be "0 0 0 0 0 0"; do not process this line.

## Output

For each collection, output "Collection #k: ", where  $k$  is the number of the test case, and then either " Can be divided " or " Can't be divided " .

Output a blank line after each test case.

## Sample Input

```
1 0 1 2 0 0
1 0 0 0 1 1
0 0 0 0 0 0
```

## Sample Output

```
Collection #1:
Can't be divided.
Collection #2:
Can be divided.
```

## Problem Source

Mid-Central European Regional Contest 1999

### 【题目大意】

Marsha 和 Bill 搜集到一些大理石块，他们想要平均分配这些石块。如果每块石头的价值都一样，那么事情显得异常简单。但是一些大理石块比较大，或更漂亮一些。因此，Marsha 和 Bill 给每块大理石分配一个价值：从数字 1~6。现在他们尝试分配石块，使得每人得到的大理石块总价值相同。

但是，他们明白就这样分配大理石块仍然是很困难的（即使所有大理石的总价值是偶数）。例如，价值为 1 的大理石块一个、价值为 3 的大理石块一个，以及价值为 4 的大理石块

两个，他们就无法把大理石块按总价值平分。因此他们请你写一个程序，判断是否能公平分配大理石块。

### 输入格式

每行输入就是一组要平分的大理石块。一行是六个正整数  $n_1, n_2, \dots, n_6$ ,  $n_i$  是价值为  $i$  的大理石块数。所以上面的例子，其输入为“1 0 1 2 0 0”。大理石块的最大数量是 20000。

当一行是“0 0 0 0 0 0”时输入结束，该行不需要处理。

### 输出格式

对每组大理石数据，输出“Collection #k:”， $k$  是测试数据组的编号，然后是“Can be divided.”或者“Can't be divided.”。

每组测试数据后输出一个空行。

### 【算法分析】

有 6 种价值的大理石块，每种价值大理石的最大数量是 20000，要求按总价值进行平分。最简单的办法是枚举，但真要枚举的话，其计算量是非常庞大的。

本题在求解时，对原始数据进行了预处理，大大降低了每种大理石的数量，然后通过枚举或者动态规划的算法，都能很快判断是否能公平分配大理石块。

#### (1) 数据预处理

将各种价值大理石的块数，存放到数组中：

```
int a[7];
```

价值为  $i$  ( $1 \leq i \leq 6$ ) 的大理石块个数为  $a[i]$  ( $0 \leq a[i] \leq 20000$ )。在平分大理石时，一种较差的情况是，一个人只需要拿一种大理石块，而另一个人拿其余的大理石块；一般情况是，两个人对每种大理石块都有划分。例如：“1 0 2 0 1 0”，一个人拿走价值为 3 的大理石块，其余归另一个人；而“2 1 1 1 1 1”，一个人只拿某一种大理石就不行了，需要拿到多种大理石。这两组数据，对任意价值的大理石块，无论增加多少个 2，仍然是可以平分的，因为他们只要平分这个 2 就可以了。

因此我们不需要对每种价值的大理石，按多达 20000 块进行平分，而只需要对一个最低限度的大理石块数  $n$  ( $0 \leq n \leq a[i]$ ) 进行平分，其余的部分，他们直接平分即可。显然  $n$  是偶数，经过在线测试， $n$  的值很小， $n=6$ 。

关键代码如下：

```
if(a[i]!=0 && a[i]%6==0) a[i] = 6;  
else a[i] = a[i]%6;
```

其中： $1 \leq i \leq 6$ 。

这样就大大压缩了原始数据的范围。在这个范围内，我们可以使用很多算法实现平分，如枚举、动态规划、搜索、回溯和分支限界算法等等。这里先介绍动态规划算法，然后介绍枚举算法的实现。

#### (2) 使用动态规划算法平分

两个人能否平分大理石块，可以简化为一个人能否从大理石块的总价值中取出一半价值的大理石块。

设大理石块总价值为  $\text{sum} = \sum(a[i]*i)$ ，若  $\text{sum}$  为奇数，是不可能实现平分的；否则令  $\text{mid}$  为大理石块总价值的一半：

```
int mid = sum/2;
```

将所有大理石的价值进行组合，可以构造出各种价值，我们只需要关心从 0~mid 的价值，即根据大理石的价值划分动态规划的阶段，存储到数组中：

```
bool visit[200];
```

因为  $a[i] \leq 6$  ( $1 \leq i \leq 6$ )， $sum \leq 6 \times (1+2+3+4+5+6) = 126$ ，则  $mid \leq 63$ 。

当  $visit[j] == true$  ( $0 \leq j \leq mid$ ) 时，表示有一个人获得价值  $j$ ，另一个人获得价值  $sum-j$  的大理石块分配方案。若  $visit[j] = false$ ，说明这种分配方案不存在。

我们的任务就是计算出  $visit[mid]$  是 true 还是 false。

显然有  $visit[0] = true$  的方案存在，即一个人什么都不分，另外一个人拿走全部的大理石。

设  $i$  ( $1 \leq i \leq 6$ ) 为大理石块的价值，若  $visit[j] = true$  ( $0 \leq j \leq mid$ )，则增加  $k$  ( $1 \leq k \leq a[i]$ ) 块大理石，该平分方案仍然是成立的。

### 【程序代码】

---

程序名称：	zju1149.c
题    目：	Dividing
提交语言：	C
运行时间：	0ms
运行内存：	160KB

---

```
#include<stdio.h>
#include<string.h>

int a[7]; //各种价值大理石的块数

//使用动态规划算法平分，形参 sum 是大理石块总价值
void DP(int sum) {
    int i,j,k;
    int mid = sum/2; //大理石块总价值的一半
    char visit[200]; //大理石的状态数组
    memset(visit,0,sizeof(visit));
    int t;
    visit[0] = 1; //初值
    //对每种价值的大理石块
    for(i=1; i<=6; i++)
        for(j=mid; j>=0; j--)
            //价值为 j 的分配方案存在
            if (visit[j])
                for(k=1; k<=a[i]; k++)
                {
                    t = j+i*k;
                    if (t>mid) break;
                    //增加 k 块价值为 i 大理石，该平分方案仍然是成立的
                    else visit[t] = 1;
                    //找到了分配方案
                    if (t==mid) {
```

```

        printf("Can be divided.\n\n");
        return;
    }
}

printf("Can't be divided.\n\n");
}

int main()
{
    int i;
    int iCase = 1;                //测试数据组的编号
    while(1)
    {
        int sum = 0;              //预处理之后大理石的总价值
        for (i=1; i<=6; i++){
            scanf("%d",&a[i]);
            //数据预处理
            if(a[i]!=0 && a[i]%6==0) a[i] = 6;
            else a[i] = a[i]%6;
            sum += a[i]*i;
        }
        if (sum==0) break;        //输入结束
        printf("Collection #d:\n", iCase++);
        //总价值为奇数，无法平分
        if(sum%2) {
            printf("Can't be divided.\n\n");
            continue;
        }
        //调用动态规划算法平分
        DP(sum);
    }
    return 0;
}

```

### 【其他算法】

枚举算法比较简单，是一个完全组合。对每一种组合，计算总价值  $mid$ ，如果  $mid = sum/2$ ，那就是找到了平分方案。看到 6 重循环，有点让人心慌，其实  $a[i] \leq 6$  ( $1 \leq i \leq 6$ )，最坏情况下的计算工作量是：

$$6^6 = 46656$$

在线测试的结果，仍然是 0ms。即实际的循环总数远远小于这个数。

```

void enumerate(int sum) {
    int mid;
    int i1,i2,i3,i4,i5,i6;
    for(i1=0; i1<=a[1]; i1++)
        for(i2=0; i2<=a[2]; i2++)

```

```

for(i3=0; i3<=a[3]; i3++)
for(i4=0; i4<=a[4]; i4++)
for(i5=0; i5<=a[5]; i5++)
for(i6=0; i6<=a[6]; i6++)
{
    mid = 1*i1+2*i2+3*i3+4*i4+5*i5+6*i6;
    if(mid==sum/2) {
        printf("Can be divided.\n\n");
        return;
    }
}
printf("Can't be divided.\n\n");
}

```

## 第七章 回溯算法题

在本章的题目主要是回溯算法题。需要使用回溯算法，实现题目的要求。

### ZJU1145-Dreisam Equations<sup>[1]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768KB

Special Judge

---

During excavations in the Dreisamwuste, a desert on some far away and probably uncivilized planet, sheets of paper containing mysterious symbols had been found. After a long investigation, the project scientists have concluded that the symbols might be parts of equations. If this were true, it would be proof that the Dreisamwuste was civilized a long long time ago.

The problem, however, is that the only symbols found on the sheets are digits, parantheses and equality signs. There is strong evidence that the people living in the Dreisamwuste knew only of three arithmetic operations: addition, subtraction, and multiplication. It is also known that the people of the Dreisamwuste did not have prioritization rules for arithmetic operations—they evaluate all terms strictly left to right. For example, for them the term  $3 + 3 * 5$  would be equal to 30, and not 18.

But currently, the sheets do not contain any arithmetic operators. So if the hypothesis is true, and the numbers on the sheets form equations, then the operators must have faded away over time.

You are the computer expert who is supposed to find out whether the hypothesis is sensible or not. For some given equations (without arithmetic operators) you must find out if there is a possibility to place +, -, and \* in the expression, so that it yields a valid equation. For example, on one sheet, the string "18=7 (5 3) 2" has been discovered. Here, one possible solution is "18=7+(5-3)\*2". But if there was a sheet containing "5=3 3", then this would mean that the Dreisamwuste people did not mean an equation when writing this.

### Input

Each equation to deal with occupies one line in the input. Each line begins with a positive integer (less than 230) followed by an equality sign =. (For your convenience, the Dreisamwuste inhabitants used equations with trivial left sides only.) This is followed by up to 12 positive integers forming the right side of the equation. (The product of these numbers will be less than 230.) There might be some parentheses around groups of one or more numbers. There will be no line containing more than 80 characters. There is no other limit for the amount of the parentheses in the equation. There will always be at least one space or parenthesis between two numbers, otherwise the

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1145>

occurrence of white space is unrestricted.

The line containing only the number 0 terminates the input, it should not be processed.

## Output

For each equation, output the line "Equation #n: ", where n is the number of the equation. Then, output one line containing a solution to the problem, i. e. the equation with the missing +, -, and \* signs inserted. Do not print any white space in the equation.

If there is no way to insert operators to make the equation valid, then output the line "Impossible ".

Output one blank line after each test case.

## Sample Input

```
18 = 7 ( 5 3 ) 2
30 = 3 3 5
18 = 3 3 5
5 = 3 3
0
```

## Sample Output

```
Equation #1:
18=7+(5-3)*2
Equation #2:
30=3+3*5
Equation #3:
Impossible
Equation #4:
Impossible
```

## Problem Source

Mid-Central European Regional Contest 1999

### 【题目大意】

在一个遥远而未开化的行星 Dreisamwuste 上有一片沙漠，发掘的出土文物中，发现了一些画有神秘符号的纸片。经过长期研究之后，该工程科学家猜测这些符号可能是等式的一部分。如果这是真实的，就能证明 Dreisamwuste 文明很久以前已经存在了。

然而，问题是纸片上的符号只有数字，括号和等号。有足够的证据可以证明 Dreisamwuste 人只懂得三种运算符的运算：加，减和乘，而且算术运算没有优先规则：他们只是严格地将每项数据从左边计算到右边。例如， $3+3\times 5$ ，他们的计算结果是 30，而不是 18。

但现在纸片上的等式里没有任何算术运算符。因此如果这些假设是正确的，而且这些数字能形成等式，那么这些运算符早已随时间而消逝了。

你是计算机专家，应该能够发现这些假设是否正确。对于某些等式（没有算术运算符），如果在表达式中重新放置运算符 +，- 和 \*，以便构造一个有效的等式。例如，在一张纸上，

有字符串“ $18=7(5\ 3)\ 2$ ”。一种可能的结果是  $18=7+(5-3)\times 2$ 。但是如果一张纸上有“ $5=3\ 3$ ”，那么在写下这个式子时，Dreisamwuste 人不认为这是一个等式。

### 输入格式

每个等式在输入中占一行。每行开头是一个正整数（小于 230），接着是“=”。（为了方便，Dreisamwuste 人在左边只用了数字。）之后是多达 12 个正整数，构成等式的右边。（这些数的乘积小于 230。）有些数可能有括号。一行不超过 80 个字符。等式中的括号数没有限制。两个数之间至少有一个空格或括号。

输入一行只有 0 时结束，不需处理。

### 输出格式

对于每个等式，输出一行 "Equation #n: "， $n$  是等式的编号。然后输出一行，是该等式的解决方案，也就是插入了+，-和 $\times$ 运算符的等式。不要在等式中输出空格。

如果等式不成立，那么输出 "Impossible "。

每组测试数据后输出一个空行。

### 【算法分析】

题目中每个等式的右边，是一些数字和括号，要求放置适当的运算符+，-和 $\times$ ，使运算结果等于左边。对于每个可以放置运算符的位置，都可以放置+，-和 $\times$ ，即解空间是一棵完全三叉树，所以时间复杂度是  $O(3^n)$ 。

本题采用回溯算法。

（1）等式数据的读取

使用数组表示等式：

```
char a[100];
```

该数组的位置指针为：

```
int apos;
```

因为不知道等式有多少项，所以每次读取一行：gets(a)。

如果字符串 a 中没有“=”，则表示输入结束。而结束行的数字“0”，担心有多余的空格符。

（2）构造伪表达式

从数组 a 中分离出数字、括号，并确定加入运算符的位置，构成伪表达式，存放到数组：

```
int b[100];
```

该数组的位置指针为：

```
int bpos;
```

对样例数据： $18=7(5\ 3)\ 2$ ，数组 b 的值如下：

下标	0	1	2	3	4	5	6	7	8
值	7	-6	-1	5	-6	3	-2	-6	2

表中阴影部分是数字，-1 表示左括号，-2 表示右括号，-6 表示该位置应该有一个运算符。为便于判断，使用数组表示运算符在数组 b 中的位置：

```
int op[30];
```

该数组的位置指针为：



```
int opos;
```

如上例所示，数组 `op` 的值如下：

下标	0	1	2
值	1	4	7

表示数组 `b` 中，位置为 1、4 和 7 的地方，是运算符。

本题构造伪表达式的过程是烦琐的。详细过程见代码中的注释。

### (3) 构造表达式

采用回溯算法构造表达式，函数是：

```
void backtrack(int dep);
```

其中形参 `dep` 表示第几个运算符。

- 如果 `dep = opos`，表示所有的运算符构造完毕

此时，表达式的结果等于左边的数 `iLeft` 时，即找到了答案。

- 如果 `dep < opos`，对第 `dep` 个运算符，可以放置 +，- 和 \*，并进入相应的子树。

### (4) 括号的处理

采用递归函数：

```
int bracket();
```

这是一个间接递归的函数。也就是说，如果一对括号中还有括号的话，则继续调用 `bracket()` 函数。也可以使用堆栈的方法实现。

## 【程序代码】

---

程序名称：	<code>zju1145.c</code>
题    目：	<code>Dreisam Equations</code>
提交语言：	<code>C</code>
运行时间：	<code>40ms</code>
运行内存：	<code>160KB</code>

---

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define LEFT -1           //左括号
#define RIGHT -2          //右括号
#define MUL -3            // *号
#define ADD -4            // +号
#define SUB -5            // -号
#define OP -6             //有运算符
#define NONE -10

char a[100];              //原始的等式数据
int b[100];               //伪表达式
int best[100];            //答案
int op[30];               //运算符在数组 b 中的位置
```

```

int bn;                //数组 b 的项数
int iLeft;             //等式左边的数
int apos, bpos, opos;
int possible;          //是否有解

//位置指针跳过空格
void space() {
    while (a[apos] && (a[apos] == ' '))
        apos++;
}

int compute();
//计算括号里面的值
int bracket() {
    int sum;
    if (b[bpos] == LEFT) {
        bpos++;                //跳过左括号
        sum = compute();        //计算括号里面的值（注意间接递归）
        bpos++;                //跳过右括号
    }
    else
        sum = b[bpos++];        //没有括号
    return sum;
}

//计算表达式
int compute() {
    int sum = bracket();        //右边第一个数
    //对每一个运算符进行计算
    while (b[bpos] == MUL || b[bpos] == ADD || b[bpos] == SUB) {
        int operation = b[bpos++];    //取出运算符
        int ret = bracket();          //取出下一个数
        //根据运算符进行计算
        switch (operation) {
            case MUL: sum *= ret; break;
            case ADD: sum += ret; break;
            case SUB: sum -= ret; break;
        }
    }
    return sum;
}

//回溯算法，构造表达式
void backtrack(int dep) {
    if (possible) return;        //得到答案

```

```

int i;
//所有的运算符构造完毕
if (dep==opos) {
    bpos = 0;
    int iRight = compute();           //右边表达式的值
    if (iRight == iLeft) {           //得到答案
        possible = 1;
        for (i=0; i<bn; i++)
            best[i] = b[i];
    }
    return;
}
//当前结点的3个孩子结点, 分别使用+, -和*运算符进行构造
b[op[dep]] = MUL; backtrack(dep+1);
b[op[dep]] = ADD; backtrack(dep+1);
b[op[dep]] = SUB; backtrack(dep+1);
}

//输出结果
void print(int *q) {
    printf("%d=", iLeft);
    int i;
    for (i=0; i<bn; i++)
        switch (q[i]) {
            case ADD: printf("+"); break;
            case MUL: printf("*"); break;
            case SUB: printf("-"); break;
            case LEFT: printf("("); break;
            case RIGHT: printf(")"); break;
            case OP: printf("?"); break;
            default: printf("%d", q[i]); break;
        }
}

int main() {
    int iCase = 0;                     //测试例编号
    int i;
    while (gets(a) && strchr(a, '=')) {
        possible = 0;
        for (i=0; i<90; i++)
            b[i] = NONE;               //初值
        apos = 0;
        sscanf(a, "%d", &iLeft);       //左边的数
        //将位置指针移到数字的后面
        while (a[apos] && isdigit(a[apos])) apos++;
    }
}

```

```

space();
apos ++;                                //跳过'='
bn = 0;
opos = 0;
while (space(), a[apos]) {
    if (a[apos] == '(') {                //左括号
        b[bn++] = LEFT;
        apos++;
        continue;
    }
    if (a[apos] == ')') {                //右括号
        b[bn++] = RIGHT;
        apos++;
    }
    else {                                //读取数字
        sscanf(a+apos, "%d", &b[bn++]);
        while (a[apos] && isdigit(a[apos])) apos++;
    }
    space();
    //如果不是结尾和')', 则有一个运算符
    if (a[apos] && a[apos] != ')') {
        op[opos++] = bn;
        b[bn++] = OP;
    }
}
//回溯算法, 构造表达式
backtrack(0);
printf("Equation # %d:\n", ++iCase);
//表达式右边只有一项
if (bn==1 && iLeft == b[0])
    printf("%d=%d\n", iLeft, iLeft);
//无解
else if (bn==0 || !possible)
    printf("Impossible\n");
else { //输出答案
    print(best);
    printf("\n");
}
printf("\n");
}
return 0;
}

```

# ZJU1157-A Plug for UNIX<sup>[1, 2, 3]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768KB

---

You are in charge of setting up the press room for the inaugural meeting of the United Nations Internet eXecutive (UNIX), which has an international mandate to make the free flow of information and ideas on the Internet as cumbersome and bureaucratic as possible.

Since the room was designed to accommodate reporters and journalists from around the world, it is equipped with electrical receptacles to suit the different shapes of plugs and voltages used by appliances in all of the countries that existed when the room was built. Unfortunately, the room was built many years ago when reporters used very few electric and electronic devices and is equipped with only one receptacle of each type. These days, like everyone else, reporters require many such devices to do their jobs: laptops, cell phones, tape recorders, pagers, coffee pots, microwave ovens, blow dryers, curling irons, tooth brushes, etc. Naturally, many of these devices can operate on batteries, but since the meeting is likely to be long and tedious, you want to be able to plug in as many as you can.

Before the meeting begins, you gather up all the devices that the reporters would like to use, and attempt to set them up. You notice that some of the devices use plugs for which there is no receptacle. You wonder if these devices are from countries that didn't exist when the room was built. For some receptacles, there are several devices that use the corresponding plug. For other receptacles, there are no devices that use the corresponding plug.

In order to try to solve the problem you visit a nearby parts supply store. The store sells adapters that allow one type of plug to be used in a different type of outlet. Moreover, adapters are allowed to be plugged into other adapters. The store does not have adapters for all possible combinations of plugs and receptacles, but there is essentially an unlimited supply of the ones they do have.

## This problem contains multiple test cases!

The first line of a multiple input is an integer  $N$ , then a blank line followed by  $N$  input blocks. Each input block is in the format indicated in the problem description. There is a blank line between input blocks.

The output format consists of  $N$  output blocks. There is a blank line between output blocks.

## Input

The input will consist of one case. The first line contains a single positive integer  $n$  ( $1 \leq n \leq$

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1157>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1087>

[3] <http://online-judge.uva.es/p/v7/753.html>

100) indicating the number of receptacles in the room. The next  $n$  lines list the receptacle types found in the room. Each receptacle type consists of a string of at most 24 alphanumeric characters. The next line contains a single positive integer  $m$  ( $1 \leq m \leq 100$ ) indicating the number of devices you would like to plug in. Each of the next  $m$  lines lists the name of a device followed by the type of plug it uses (which is identical to the type of receptacle it requires). A device name is a string of at most 24 alphanumeric characters. No two devices will have exactly the same name. The plug type is separated from the device name by a space. The next line contains a single positive integer  $k$  ( $1 \leq k \leq 100$ ) indicating the number of different varieties of adapters that are available. Each of the next  $k$  lines describes a variety of adapter, giving the type of receptacle provided by the adapter, followed by a space, followed by the type of plug.

## Output

A line containing a single non-negative integer indicates the smallest number of devices that cannot be plugged in.

## Sample Input

```
1
4
A
B
C
D
5
laptop B
phone C
pager B
clock B
comb X
3
B X
X A
X D
```

## Sample Output

```
1
```

## Problem Source

East Central North America 1999

### 【题目大意】

你负责为联合国互联网管理委员会（UNIX）的首次会议准备新闻发布室，UNIX 有个国际性任务，就是使互联网上累赘和官僚的信息和想法尽可能畅通无阻。

因为新闻发布室要接待来自世界各地的记者们，所以在建造发布室时，它配备了各种电插座以满足当时各个国家电器不同形状的电插头和电压。不幸的是，发布室是在很多年以前建造的，当时记者们使用电器的种类很少，每种电器也只有一种类型的插座。现在，像其他人一样，记者们需要很多这样的设备来做他们的工作：笔记本电脑，手机，录音机，寻呼机，咖啡壶，微波炉，吹风机，卷发钳，（电动）牙刷等。当然，其中很多设备可以使用电池，但由于这次会议可能时间很长而且乏味，你希望能够提供尽可能多的插座。

在开会之前，你收集了记者们喜欢使用的各种设备，并设法安装使用。你注意到，一些设备使用的插头在发布室是没有相应插座的。你感到很好奇，外国的这些设备在建造发布室时是不是不存在的。有些插座，好多设备的插头可以使用；而另外一些插座，没有一个设备的插头能够使用。

为了设法解决这个问题，你访问了附近的一家电器供应商店。这个商店出售适配器，适配器把一种类型的插头转换为不同类型插座。此外，适配器可以插入其他的适配器。这家商店没有用于所有可能组合的插头和插座的适配器。所有在售的适配器，他们基本上是无限量供应。

### 本题包含多组测试例！

多组测试例的第一行是一个整数  $N$ ，然后是一个空行，接着是  $N$  个输入数据块。每个数据块的格式在问题描述中给出。每个数据块之间有一个空行。

输出格式包括  $N$  个输出数据块，每个输出数据块之间有一个空行。

### 输入格式

输入只有一个测试例（指每个数据块内）。第一行是一个正整数  $n$  ( $1 \leq n \leq 100$ )，表示发布室里插座的数量。接下来  $n$  行，是在发布室里找到的插座类型。每个插座类型是一个不超过 24 个字母数字的字符串。下一行是一个正整数  $m$  ( $1 \leq m \leq 100$ )，表示你想要插入的设备数量。接下来  $m$  行，每行列出设备的名称和它使用的插头类型（与它所需要的插座类型相一致）。设备名称是一个不超过 24 个字母数字的字符串。没有两个设备的名字恰好相同。插头类型和设备名称之间有一个空格。下一行是一个正整数  $k$  ( $1 \leq k \leq 100$ )，表示可用的不同类型的适配器数量。接下来  $k$  行，每行描述一种类型的适配器：适配器提供的插座类型，一个空格，接着是插头的类型。

### 输出格式

输出一行，是一个正整数，表示不能插入的设备的最小数。

### 【算法分析】

给定插头和插座的类型和数量，再给定适配器的类型，而适配器的数量是无限的，且适配器可以插入其他的适配器，计算没有找到插座的插头数量。

代码参考自标程 (<http://plg1.cs.uwaterloo.ca/~acm00/regionals/>)。

这里选择的是回溯算法，标程中还有使用二分图的算法，读者可以参考该网站。

#### (1) 数据结构

插座的数量

```
int numReceps;
```

插头的数量

```
int numPlugs;
```

插头的结构体：

```
struct plug {
```

```

        int recep;
        bool list[400];
    } plugs[100];

```

变量 **recep** 是插头需要的插座类型，数组 **list** 是使用适配器后，可以匹配的插座类型。  
适配器的结构体：

```

    struct adapter {
        int recep, plug;
    } adapts[100];

```

变量 **recep**, **plug** 是插座和插头的类型。

插座和插头的类型是字符串，需要映射成数字，由函数完成：

```

    int findName(string name);

```

形参 **name** 是插座或插头的类型，返回值就是映射后的数字。对样例数据，映射关系如下：

下标	0	1	2	3	4
插座或插头名称	A	B	C	D	X

每个电器的插头需要的插座类型，映射关系如下：

下标（电器序号）	0	1	2	3	4
插座类型号	1	2	1	1	4

适配器的映射关系如下：

下标	字母表示		数字映射	
	插座	插头	插座	插头
0	B	X	1	4
1	X	A	4	0
2	X	D	4	3

(2) 每个插头使用适配器后，可以匹配的插座类型

计算每一个插头，在使用适配器后可以匹配的插座类型：

```

    for(int i=0; i<numPlugs; i++)
        useAdapter(i);

```

计算匹配的功能由函数完成：

```

    void useAdapter(int i);

```

形参 **i** 是插头的编号。

程序中匹配的结果存储在相应插头的数组 **list** 中。由于适配器可以插入其他的适配器，所以使用一个适配器后，又要重新搜索一遍。

对样例数据，计算结果如图 7-1 所示：

表中数据表示，使用适配器后，每个电器可以使用的插座编号。例如电器 2，可以使用插座 0, 1, 3 和 4，即插座 A, B, D 和 X。从表中看出，每个电器都能找到相应的插座，但本题是 5 个电器只有 4 个插座，就有 1 个电器找不到插座了。



		插座编号				
		0	1	2	3	4
电 器 编 号	0	1	1	0	1	1
	1	0	0	1	0	0
	2	1	1	0	1	1
	3	1	1	0	1	1
	4	1	0	0	1	1

图 7-1 插头使用适配器后可以匹配的插座类型

### (3) 建立插头与插座的对应关系

使用辅助数组帮助匹配关系的建立。插头与插座匹配的情况：

```
int matchPlug[100];
```

插座与插头匹配的情况，这是上一个数组的反函数：

```
int matchRecep[100];
```

对每一个插头，都去找与之匹配的插座：

```
int i=0;
while(i < numPlugs)
{
    .....
    if (matchPlug[i] == -1 && backtrack(i)) i = 0;
    else i++;
}
```

如果该插头已经匹配（即找到了插座），就找下一个插头（ $i++$ ）；如果该插头还没有匹配，就通过回溯算法寻找匹配的插座，因为改变了匹配状态，需要重新开始搜索（ $i=0$ ）。

回溯算法的实现由函数完成：

```
bool backtrack(int node);
```

形参 `node` 表示当前插头。回溯算法的实现细节请参考代码中的注解。

计算结束时，数组 `matchPlug` 的数值如下：

下标	0	1	2	3	4
插头匹配的插座	3	2	1	0	-1

例如在上表中，插头 0 插入插座 3 中，插头 1 插入插座 2 中，等等。

数组 `matchRecep` 的数值如下：

下标	0	1	2	3	4
插座匹配的插头	3	2	1	0	-1

例如在上表中，插座 0 匹配插头 3，插座 1 匹配插头 2，等等。显然这是上表的反函数。

### (4) 统计哪些插头没有找到插座

从数组 `matchPlug` 中可以看出，值为 -1 的插头没有找到插座。将所有值为 -1 的个数统计出来即可。

## 【程序代码】

---

程序名称: zju1157.cpp  
题 目: A Plug for UNIX  
提交语言: C++  
运行时间: 0ms  
运行内存: 228KB

---

```
#include <iostream>
using namespace std;

string names[400];           //插头和插座的名称
int numName;                 //插头和插座的编号
int numReceps;               //插座的数量
int numAdapts;               //适配器的数量
int matchPlug[100];          //插头与插座匹配的情况
int matchRecep[100];         //插座与插头匹配的情况
bool canUse[100];

//插头的结构体
struct plug {
    int recep;                //需要的插座类型
    bool list[400];           //使用适配器后,可以匹配的插座类型
} plugs[100];

//适配器的结构体
struct adapter {
    int recep, plug;          //插座和插头的类型
} adapts[100];

//将字符串映射的数字
int findName(string name)
{
    for(int i=0; i<numName; i++)
        if (name == names[i]) return i;    //该字符串已经在数组中
    names[numName++] = name;                //新增字符串
    return numName-1;
}

//插头 i 使用适配器后,可以匹配的插座类型
void useAdapter(int i)
{
    //插头 i 本身需要的插座
    plugs[i].list[plugs[i].recep] = true;
    //由于适配器可以插入其他的适配器,所以使用一个适配器后,又要重新搜索
    bool flag;
```

```

do {
    flag = false;
    //搜索所有的适配器
    for(int j=0; j<numAdapts; j++) {
        if (plugs[i].list[adapts[j].recep] &&
            !plugs[i].list[adapts[j].plug])
        {
            flag = true;
            plugs[i].list[adapts[j].plug] = true;
        }
    }
    //只要找到了一个适配器 (flag = true)，就要重新搜索一遍
} while (flag);
}

//回溯算法的实现，插头 node 和所有的插座匹配，是完全 numReceps 叉树
bool backtrack(int node)    {
    //对所有的插座搜索
    for(int j=0; j<numReceps; j++) {
        //插头 node 与插座 j 匹配，且插座 j 可用
        if (plugs[node].list[j] && canUse[j])
        {
            if (matchRecep[j] == -1) {                //插座 j 尚未匹配
                //建立插头 node 与插座 j 的匹配关系
                matchPlug[node] = j;
                matchRecep[j] = node;
                return true;
            }
            else
            {
                //插座 j 尚未匹配，保存其匹配状态，换成插头 node
                int save = matchRecep[j];
                matchRecep[j] = node;
                matchPlug[save] = -1;
                matchPlug[node] = j;
                canUse[j] = false;
                //回溯，判断插头 matchRecep[j] 能否插入其他插座
                if (backtrack(save))
                    return true;
                //恢复回溯前的状态
                canUse[j] = true;
                matchPlug[node] = -1;
                matchPlug[save] = j;
                matchRecep[j] = save;
            }
        }
    }
}

```

```

        }
    }
}
return false;
}

int main() {
    int numPlugs;           //插头的数量
    string name;
    int N;                  //数据块的数量
    scanf("%d", &N);
    while (N--)
    {
        //读取插座的名称，建立名称数组
        cin >> numReceps;
        for(int i=0; i<numReceps; i++)
            cin >> names[i];
        numName = numReceps;
        //读取插头的名称，建立插头数组。电器的名称是没有用的
        cin >> numPlugs;
        for(int i=0; i<numPlugs; i++)
        {
            cin >> name >> name;
            plugs[i].recep=findName(name);
            memset(plugs[i].list, false, sizeof(plugs[i].list));
        }
        //读取适配器的名称，建立适配器数组
        cin >> numAdapts;
        for(int i=0; i<numAdapts; i++)
        {
            cin >> name;
            adapts[i].recep = findName(name);
            cin >> name;
            adapts[i].plug = findName(name);
        }
        //每个插头使用适配器后，可以匹配的插座类型
        for(int i=0; i<numPlugs; i++)
            useAdapter(i);
        memset(matchPlug, 0xff, sizeof(matchPlug));
        memset(matchRecep, 0xff, sizeof(matchRecep));
        //建立插头与插座的对应关系，匹配数组是 matchPlug
        int i=0;
        while(i < numPlugs)
        {
            for(int j=0; j<numReceps; j++)

```

```

        canUse[j] = true;
        if (matchPlug[i] == -1 && backtrack(i)) i = 0;
        else i++;
    }
    //统计哪些插头没有找到插座
    int min = 0; //没有找到插座的插头数量
    for(int i=0; i<numPlugs; i++)
        if (matchPlug[i] == -1) min++;
    cout << min << endl;
    if (N) printf("\n");
}
return 0;
}

```

## 第八章 图论算法题

在本章的题目主要是图论算法题。需要使用图论算法，实现题目的要求。

### ZJU1082-Stockbroker Grapevine<sup>[1、2、3]</sup>

---

Time limit: 1 Second    Memory limit: 32768KB

---

Stockbrokers are known to overreact to rumours. You have been contracted to develop a method of spreading disinformation amongst the stockbrokers to give your employer the tactical edge in the stock market. For maximum effect, you have to spread the rumours in the fastest possible way.

Unfortunately for you, stockbrokers only trust information coming from their "trusted sources". This means you have to take into account the structure of their contacts when starting a rumour. It takes a certain amount of time for a specific stockbroker to pass the rumour on to each of his colleagues. Your task will be to write a program that tells you which stockbroker to choose as your starting point for the rumour, as well as the time it will take for the rumour to spread throughout the stockbroker community. This duration is measured as the time needed for the last person to receive the information.

#### Input

Your program will input data for different sets of stockbrokers. Each set starts with a line with the number of stockbrokers. Following this is a line for each stockbroker which contains the number of people who they have contact with, who these people are, and the time taken for them to pass the message to each person. The format of each stockbroker line is as follows: The line starts with the number of contacts ( $n$ ), followed by  $n$  pairs of integers, one pair for each contact. Each pair lists first a number referring to the contact (e.g. a "1" means person number one in the set), followed by the time in minutes taken to pass a message to that person. There are no special punctuation symbols or spacing rules.

#### Output

For each set of data, your program must output a single line containing the person who results in the fastest message transmission, and how long before the last person will receive any given message after you give it to this person, measured in integer minutes.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1082>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1125>

[3] <http://acm.uva.es/archive/nuevoportal/data/problem.php?p=2241>

Each person is numbered 1 through to the number of stockbrokers. The time taken to pass the message on will be between 1 and 10 minutes (inclusive), and the number of contacts will range between 0 and one less than the number of stockbrokers. The number of stockbrokers will range from 1 to 100. The input file is terminated by a set of stockbrokers containing 0 (zero) people.

It is possible that your program will receive a network of connections that excludes some persons, i.e. some people may be unreachable. If your program detects such a broken network, simply output the message " disjoint ". Note that the time taken to pass the message from person A to person B is not necessarily the same as the time taken to pass it from B to A, if such transmission is possible at all.

## SAMPLE INPUT

```
3
2 2 4 3 5
2 1 2 3 6
2 1 2 2 2
5
3 4 4 2 8 5 3
1 5 8
4 1 6 4 10 2 7 5 2
0
2 2 5 1 5
0
```

## SAMPLE OUTPUT

```
3 2
3 10
```

## Problem Source

South Africa 2001

### 【题目大意】

股票经纪人以对传闻的而闻名。你已签定一个合同，就是研究一种方法，在股票经纪人中间传播假情报，以便雇主在股票市场中赢得先机。为了得到最大效益，你必须尽可能快地散布传闻。

不幸的是，股票经纪人只信任来自他们“可靠来源”的消息。这意味着，在散播传闻时，你必须考虑他们所接触的人际关系。某个股票经纪人需要花一些时间把传闻传递给他的每一位同事。你的任务是写一个程序，确定从哪个证券经纪人开始散播传闻，以及传闻传递到整个证券界时所需要的时间。这个时间是指最后一个人收到传闻的时间。

### 输入格式

程序输入有多组不同的股票经纪人。每组数据开始的一行，是股票经纪人的数目。接下来每个股票经纪人一行：①这个股票经纪人的联系人个数；②他们的编号；③每个人传递传闻所需要的时间。输入格式如下：行首是联系人个数( $n$ )，接着是  $n$  对整数，每对整数对应一位联

系人。每对数的第一个是联系人编号（例如，“1”是该组中第一号人物），然后是此人传递消息的时间（分钟）。数据之间没有特殊的标点符号或空格。

**输出格式**

对每组数据，程序输出一行，包含：①消息传递最快的人；②最后一个人得知消息所需要的时间（分钟），以整数表示。

每个人从 1 开始编号，一直到股票经纪人的总数。传递消息的时间在 1~10（含）分钟之间。联系人的个数在 0 到小于股票经纪人的总数之间。股票经纪人有 1~100 人。当股票经纪人的个数是 0 时，输入结束。

程序很可能会接收到一个不完整的人际关系网络，也就是某些人不能收到消息。如果程序检测到一个不完整的网络，只需输出“disjoint”。注意：假如某一对股票经纪人之间可以互相传递消息，那么从 A 传给 B 的时间并不等于从 B 传给 A 的时间。

**【算法分析】**

本题需要计算每个股票经纪人把消息传递给其他所有股票经纪人所需的时间，然后从中查找传递时间最小的那个股票经纪人，以及他所花费的时间。只要有一个股票经纪人没有传递到，就说明人际关系网络是不完整的。这需要采用 Floyd 方法，关于 Floyd 方法的具体描述请见 ZJU1053-FDNY to the Rescue!在一般的数据结构和计算方法的书籍中都有。

**（1）样例分析**

样例数据 1:

人际关系网络如表 8-1 所示。经过 Floyd 算法之后，原有数据没有变化。第 3 号股票经纪人把消息传递出去的时间最短，是 2min。

表 8-1 样例数据 1 的人际关系网络

0	4	5
2	0	6
2	2	0

样例数据 2:

人际关系网络如表 8-2 所示。

经过 Floyd 算法之后，人际关系及消息传递的数据如表 8-3 所示。

表 8-2 样例数据 2 的人际关系网络

0	8	0	4	3
0	0	0	0	8
6	7	0	10	2
0	0	0	0	0
5	5	0	0	0

表 8-3 样例数据 2，经过 Floyd 算法之后的人际关系网络

0	8	0	4	3
13	0	0	17	8
6	7	0	10	2
0	0	0	0	0
5	5	0	9	0

可以看出，只有第 3 号股票经纪人能够把消息传递给所有其他的股票经纪人，所需要的时间是 10min。



## (2) 数据结构

使用数组  $g$  存储人际关系:

```
int g[100][100];
```

股票经纪人的数量为

```
int n;
```

## (3) Floyd 算法的实现

由函数 `Floyd()` 完成。

## (4) 查找传递时间最小的股票经纪人, 及所花费的时间

首先必须确保某股票经纪人能够把消息传递给所有其他的股票经纪人, 然后在这些股票经纪人中间, 查找传递时间最小的那个股票经纪人, 以及他所花费的时间。

## 【程序代码】

---

程序名称:	zju1082.c
题    目:	Stockbroker Grapevine
提交语言:	C
运行时间:	00:00.00
运行内存:	428K

---

```
#include <stdio.h>
#include <memory.h>

int g[100][100];           //存储人际关系
int n;                     //股票经纪人的数量

//实现 Floyd 算法
void floyd() {
    int i, j, k;
    for(k=0; k<n; k++)
        for(i=0; i<n; i++)
            if (g[i][k]!=0)                // (i, k) 之间有路径
                for(j=0; j<n; j++)
                    if (g[k][j] && i!=j)    // (k, j) 之间有路径
                        if((g[i][j]==0)|| (g[i][j]>g[i][k]+g[k][j]))
                            g[i][j]=g[i][k]+g[k][j];
}

int main()
{
    int i, j;
    //股票经纪人联系人的个数, 联系人编号和传递消息所花费的时间
    int contact, number, time;
    while(scanf("%d", &n) && n)
```

```

{
    memset(g, 0, sizeof(g));
    //读取人际关系网络
    for(i=0; i<n; i++)
    {
        scanf("%d", &contact);
        for(j=0; j<contact; j++)
        {
            scanf("%d %d", &number, &time);
            g[i][--number] = time;
        }
    }
    //调用 Floyd 算法
    floyd();
    //查找传递时间最小的股票经纪人, 及所花费的时间
    int min = 30000;
    number = -1;
    //对每一个股票经纪人
    for(i=0; i<n; i++)
    {
        //计算第 i 个股票经纪人传递消息的最大时间 time
        time = 0;
        for(j=0; j<n; j++)
            if (i!=j)
                if (time<g[i][j]) time = g[i][j];
                //只要有一个股票经纪人没有传递到消息
                else if(g[i][j]==0){
                    time = 30000;          //失败标记
                    break;
                }
        if (min>time)                //更新最短时间
        {
            number = i;                //最短时间的股票经纪人编号
            min = time;
        }
    }
    if (min<30000) printf("%d %d\n", (++number), min);
    else printf("disjoint\n");
}
return 0;
}

```

# ZJU1083-Frame Stacking<sup>[1、2、3]</sup>

Time limit: 1 Second    Memory limit: 32768K

Consider the following 5 picture frames placed on an  $9 \times 8$  array (Figure 8-1).

Now place them on top of one another starting with 1 at the bottom and ending up with 5 on top.

If any part of a frame covers another it hides that part of the frame below.

Viewing the stack of 5 frames we see the following (Figure 8-2).

.....	.....	.....	.....	.CCC....
EEEEEE..	.....	.....	..BBBB..	.C.C....
E...E..	DDDDDD..	.....	.B..B..	.C.C....
E...E..	D...D..	.....	.B..B..	.CCC....
E...E..	D...D..	...AAAA	.B..B..	.....
E...E..	D...D..	...A..A	..BBBB..	.....
E...E..	DDDDDD	...A..A	.....	.....
E...E..	.....	...AAAA	.....	.....
EEEEEE..	.....	.....	.....	.....
1	2	3	4	5

Figure 8-1

```
.CCC....
ECBCBB..
DCBCDB..
DCCC.B..
D.B.ABAA
D.BBBB.A
DDDDAD.A
E...AAAA
EEEEEE..
```

Figure 8-2

In what order are the frames stacked from bottom to top? The answer is EDABC.

Your problem is to determine the order in which the frames are stacked from bottom to top given a picture of the stacked frames. Here are the rules:

1. The width of the frame is always exactly 1 character and the sides are never shorter than 3 characters.
2. It is possible to see at least one part of each of the four sides of a frame. A corner shows two sides.
3. The frames will be lettered with capital letters, and no two frames will be assigned the same letter.

## INPUT DATA

Each input block contains the height,  $h$  ( $h \leq 30$ ) on the first line and the width  $w$  ( $w \leq 30$ ) on the second. A picture of the stacked frames is then given as  $h$  strings with  $w$  characters each.

## Example input

```
9
8
.CCC....
```

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1083>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1128>

[3] <http://acm.uva.es/archive/nuevportal/data/problem.php?p=2241>

```
ECBCBB..
DCBCDB..
DCCC.B..
D.B.ABAA
D.BBBB.A
DDDDAD.A
E...AAAA
EEEEEE..
```

Your input may contain multiple blocks of the format described above, without any blank lines in between. All blocks in the input must be processed sequentially.

## OUTPUT DATA

Write the solution to the standard output. Give the letters of the frames in the order they were stacked from bottom to top. If there are multiple possibilities for an ordering, list all such possibilities in alphabetical order, each one on a separate line. There will always be at least one legal ordering for each input block. List the output for all blocks in the input sequentially, without any blank lines (not even between blocks).

## Example Output

```
EDABC
```

## Problem Source

South Africa 2001

### 【题目大意】

考虑以下 5 个放置在  $9 \times 8$  阵列上的框架如图 8-1 所示。

现在依次将这 5 个框架叠在一起，第 1 个框架在底下，第 5 个框架在上面。如果上层框架的任何部份覆盖了下层框架的某部分，那么下层的这部分框架就隐藏掉了。

仔细观察下面已经叠好的 5 个框架（如图 8-2 所示）。

从下到上，这些框架是以什么顺序相叠的？答案是：EDABC。

你的问题是，从一幅已经叠好的框架图中，编程确定从下往上各个框架相叠的顺序。以下是规则：

- ① 框架中的宽度保证为 1 个字符大小，边长不少于 3 个字符长。
- ② 至少可以见到框架中每条边的一部分。一个角表示两条边。
- ③ 框架由大写字母表示，不会把同一个字母分配给两个框架。

### 输入数据

每组输入数据，第一行是框架高度  $h(h \leq 30)$ ，第二行是宽度  $w(w \leq 30)$ ，接着给出已经叠好的框架图，是  $h$  个字符串，每个字符串有  $w$  个字符。

输入有多组数据，每组数据的内容都如上所述。它们之间没有空行，而且应该按输入的顺序依次处理。

## 输出数据

编写程序,实现标准输出。按从下往上的顺序,输出各个相叠框架的字母。假如问题有多种解答,那么以字母顺序输出所有的解答,每个解答占一行。每组输入至少有一个有效解。依输入顺序输出解答,解答之间没有任何空行(包括组与组之间)。

### 【算法分析】

本题要求按从下往上的顺序,输出各个相叠框架的字母,这就要搜索各个框架之间的关系,常用的算法是拓朴排序。要实现拓朴排序,需要事先构造关系矩阵。构造关系矩阵时,必须明确各个框架的位置。

(1) 确定各个框架的位置

框架的数据结构

```
struct corner{
    int x1,y1;           //顶点坐标
    int x2,y2;
    corner() { x1 = 40, y1 = 40, x2 = -40, y2 = -40; }
};
```

使用数组 `frame` 保存所有的框架:

```
corner frame[26];
```

将框架的名称(用大写字母的序号表示)保存到数组:

```
int used[26];
```

要获得框架的实际大小,只需根据框架字母的位置,将现有框架的尺寸尽可能地向外扩展(相当于求最大最小值)。由于题目确保至少可以见到框架中每条边的一部份,所以框架的位置是唯一的。

对于样例数据,其框架的顶点坐标如下:

	x1	y1	x2	y2
A	4	4	7	7
B	1	2	5	5
C	0	1	3	3
D	2	0	6	5
E	1	0	8	5

(2) 构造拓朴排序的关系矩阵

数组 `g` 保存拓朴排序的关系矩阵为

```
char g[26][26];
```

数组 `g1` 标记已经统计的边为

```
char g1[26][26];
```

数组 `number` 统计结点作为子结点出现的次数为

```
int number[26];
```

在框架  $i$  ( $i=0\sim 25$ ) 的 4 条边上进行搜索,如果发现其他框架  $x$ ,则这两个框架之间就有一条边,也就是说它们之间有关系。所有这些关系的个数 `edges`,保存到数组 `g` 中该框架所在

行的 0 单元为

```
g[i][0] = edges;
```

同时累计框架  $x$  作为子结点的次数为

```
number[x] ++;
```

对于样例数据，其关系矩阵  $g$  如下：

	0	1	2	3	4
0	1	1	0	0	0
1	1	2	0	0	0
2	0	0	0	0	0
3	3	1	2	0	0
4	4	1	3	0	2

从表中看到，第 2 行数据全是 0，也就是框架 C 的 4 条边均是可见的。

数组 `number` 的值如下：

下标	0	1	2	3	4
值	2	3	3	1	0

从表中看到，`number[4]=0`，说明框架 E 没有作为其他结点的子结点，是父结点。

(3) 拓扑排序，并输出结果

这是一个经典的算法，属于离散数学的偏序关系。算法的详细描述，请参看离散数学知识。

算法由函数 `topSort()` 实现：

```
void topSort(int depth, int count)
```

形参 `depth` 是递归搜索的深度，`count` 是框架的数量

当搜索的深度 `depth` 等于框架的数量 `count` 时，表示全部框架搜索完毕，构造了全部的关系，则输出结果。

对于当前框架  $i$  ( $i=0\sim 25$ )，如果是父结点 (`number[i]=0`)，则加入到答案中；并与该框架相邻的所有子结点数减 1：

```
edges = g[i][0];
```

```
for(j=1; j<=edges; j++)
```

```
number[g[i][j]]--;
```

接着递归搜索下一个框架。

(4) 特殊实例

考虑如下数据：

```
5 5
AAAAA
ABBBA
AB BA
ABBBA
AAAAA
```

该数据只有两个框架，而且边全是可见的，关系矩阵  $g$  数组和子结点的次数 `number` 数组中所有的单元全为 0。采用拓朴排序搜索时，`number[0]=0` 是第一入口，`number[1]=0` 是第二入口，这时依次得到两个答案：

AB

BA

## 【程序代码】

---

程序名称:	zju1083.cpp
题    目:	Frame Stacking
提交语言:	C++
运行时间:	00:00.00
运行内存:	440K

---

```
#include <stdio.h>
#include <string.h>

//框架的数据结构
struct corner{
    int x1,y1;                //顶点坐标
    int x2,y2;
    corner() { x1 = 40, y1 = 40, x2 = -40, y2 = -40; }
};

char gg[30][30];              //已经叠好的框架图
char solution[26];            //解答
char g[26][26];               //拓朴排序的关系矩阵
char g1[26][26];              //标记已经统计的边
int number[26];               //统计结点作为子结点出现的次数
int used[26];                 //已经发现的框架字母

//增加新的边 i→j
void addEdge(int i, int j, int &edges)
{
    if(g1[i][j]==0)           //该边还没有统计过
    {
        g1[i][j] = 1;         //标记该边已经统计
        g[i][++edges] = j;     //增加一条边 i→j
        number[j] ++;         //结点 j 作为子结点出现的次数
    }
}

//拓朴排序，并输出结果
//形参 depth 是搜索的深度，count 是框架的数量
void topSort(int depth, int count)
{
    int i, j, edges;
    //全部框架都已经搜索完毕
    if(depth>=count)
```

```

{    //输出答案
    for(i=0; i<count; i++)
        printf("%c", solution[i]+'A');
    printf("\n");
    return;
}
for(i=0; i<26; i++)
    //如果有该字母所表示的框架
    if (used[i])
        该结点不是其他结点的子结点
        if (number[i]==0)
        {
            //这就是第 depth 个框架
            solution[depth++] = i;
            number[i]--;
            edges = g[i][0];
            //与该框架相邻的所有子结点个数减 1
            for(j=1; j<=edges; j++)
                number[g[i][j]]--;
            //递归搜索下一个框架
            topSort(depth, count);
            //恢复现场
            for(j=1; j<=edges; j++)
                number[g[i][j]]++;
            number[i] ++;
            depth --;
        }
}

//构造拓朴排序的关系矩阵
int build(corner frame[])
{
    memset(g, 0, sizeof(g));
    memset(g1, 0, sizeof(g1));
    memset(number, 0, sizeof(number));
    int i, j;
    int count = 0;                                //框架的数量
    for(i=0; i<26; i++)
        //如果有该字母所表示的框架
        if(used[i])
        {
            count++;
            //关系图中, 与该框架相邻的边的数量
            int edges = 0;                            //与该框架相邻的边的数量

```



```

        int x1 = frame[i].x1;           //当前框架的顶点坐标
        int y1 = frame[i].y1;
        int x2 = frame[i].x2;
        int y2 = frame[i].y2;
        //在框架的水平方向, 如果发现其他框架, 则构成关系的边
        for(j=x1; j<=x2; j++)
        {
            if(gg[j][y1]!=i)           //顶边框内
                addEdge(i, gg[j][y1], edges);
            if(gg[j][y2]!=i)           //底边框内
                addEdge(i, gg[j][y2], edges);
        }
        //在框架的垂直方向, 如果发现其他框架, 则构成关系的边
        for(j=y1; j<=y2; j++)
        {
            if(gg[x1][j]!=i)           //左边框内
                addEdge(i, gg[x1][j], edges);
            if(gg[x2][j]!=i)           //右边框内
                addEdge(i, gg[x2][j], edges);
        }
        g[i][0] = edges;
    }
    return count;
}

int main()
{
    int i, j;
    //框架的高度和宽度
    int n, m;
    char line[30], c;
    while (scanf("%d", &n)!=EOF)
    {
        memset(used, 0, sizeof(used));
        //读取框架的原始数据
        scanf("%d\n", &m);
        corner frame[26];
        for(i=0; i<n; i++) {
            gets(line);           //每次读取一行
            for(j=0; j<m; j++)
            {
                c = line[j] - 'A';
                gg[i][j] = c;     //已经叠好的框架图
                if(c!=('.')-'A')) //遇到字母

```

```

        {
            used[c] = 1;           //已经发现的框架字母
            //根据该框架字母的位置，将框架扩展到实际大小
            if(frame[c].x1>i)    frame[c].x1 = i;
            if(frame[c].y1>j)    frame[c].y1 = j;
            if(frame[c].x2<i)    frame[c].x2 = i;
            if(frame[c].y2<j)    frame[c].y2 = j;
        }
    }
    //构造拓扑排序的关系矩阵
    int count = build(frame);
    //拓朴排序，并输出结果
    topSort(0, count);
}
return 0;
}

```

## ZJU1092-Arbitrage<sup>[1、2、3]</sup>

---

Time limit: 1 Second    Memory limit: 32768K

---

Arbitrage is the use of discrepancies in currency exchange rates to transform one unit of a currency into more than one unit of the same currency. For example, suppose that 1 US Dollar buys 0.5 British pound, 1 British pound buys 10.0 French francs, and 1 French franc buys 0.21 US dollar. Then, by converting currencies, a clever trader can start with 1 US dollar and buy  $0.5 \times 10.0 \times 0.21 = 1.05$  US dollars, making a profit of 5 percent.

Your job is to write a program that takes a list of currency exchange rates as input and then determines whether arbitrage is possible or not.

### Input Specification

The input file will contain one or more test cases. On the first line of each test case there is an integer  $n$  ( $1 \leq n \leq 30$ ), representing the number of different currencies. The next  $n$  lines each contain the name of one currency. Within a name no spaces will appear. The next line contains one integer  $m$ , representing the length of the table to follow. The last  $m$  lines each contain the name  $c_i$  of a source currency, a real number  $r_{ij}$  which represents the exchange rate from  $c_i$  to  $c_j$  and a name  $c_j$  of the destination currency. Exchanges which do not appear in the table are impossible.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1092>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2240>

[3] <http://acm.uva.es/problemset/v4/436.html>

Test cases are separated from each other by a blank line. Input is terminated by a value of zero (0) for  $n$ .

## Output Specification

For each test case, print one line telling whether arbitrage is possible or not in the format "Case case: Yes" respectively "Case case: No".

## Sample Input

```
3
USDollar
BritishPound
FrenchFranc
3
USDollar 0.5 BritishPound
BritishPound 10.0 FrenchFranc
FrenchFranc 0.21 USDollar

3
USDollar
BritishPound
FrenchFranc
6
USDollar 0.5 BritishPound
USDollar 4.9 FrenchFranc
BritishPound 10.0 FrenchFranc
BritishPound 1.99 USDollar
FrenchFranc 0.09 BritishPound
FrenchFranc 0.19 USDollar

0
```

## Sample Output

```
Case 1: Yes
Case 2: No
```

## Problem Source

University of Ulm Local Contest 1996

### 【题目大意】

套汇是指货币交易中，利用不同货币的汇率差价而获得收益的一种交易。例如，假设 1 美元可兑换 0.5 英镑，1 英镑能兑换 10.0 法郎，1 法郎能兑换 0.21 美元。因此，聪明的商人能通过这种交易获利，1 美元能赚取 5% 的利润， $1 \times 0.5 \times 10 \times 0.21 = 1.05$  美元。

编程任务：读取汇率列表，确定套汇能否实现。

## 输入格式

输入包含一组或多组测试例。对每组测试例，第一行是一个整数  $n$  ( $1 \leq n \leq 30$ )，表示货币的种类。接下来  $n$  行，每行是一种货币名称（中间没有空格）。接下来一行是一个整数  $m$ ，表示下面货币兑换表格的长度。接下来  $m$  行，格式为： $c_i, r_{ij}, c_j$ ，表示所持的货币  $c_i$ ，实数  $r_{ij}$  是货币  $c_i$  与  $c_j$  的汇率，所需交换的货币  $c_j$ 。表中没有出现的汇率是不能直接兑换的。

每组测试例之间有一个空行。当  $n$  为零时，表示输入结束。

## 输出格式

对每组测试例，输出一行，套汇够实现时输出“Case case: Yes”，否则输出“Case case: No”，其中 case 是测试例编号。

## 【算法分析】

本题计算套汇是否能够实现。在所给定的货币汇率中，从任意一种货币开始兑换，只要赢利就行。这就要计算一种货币，经过各种可能的兑换之后，再兑换回自己时所获得的最好赢利。显然，解决该问题的最好算法是 Floyd 方法。

### (1) 建立邻接矩阵

首先要处理的是货币名称，将货币名称建立一个数组，即

```
char name[30][300];
```

在读取汇率数据时，每读取一个名称，就到数组 name 中查找该名称对应的序号，就实现了字符串与数字序号之间的转换。

汇率数据是单向的，也就是有向图。对样例数据 1，其邻接矩阵如下面左表所示。

对样例数据 2，其邻接矩阵如下面右表所示。

	USDollar	BritishPound	FrenchFranc
USDollar	0.00	0.50	0.00
BritishPound	0.00	0.00	10.00
FrenchFranc	0.21	0.00	0.00

	USDollar	BritishPound	FrenchFranc
USDollar	0.00	0.50	4.90
BritishPound	1.99	0.00	10.00
FrenchFranc	0.19	0.09	0.00

用数组  $g$  表示邻接矩阵：

```
double g[30][30];
```

### (2) Floyd 算法的实现

Floyd 算法的具体描述请见 ZJU1053-FDNY to the Rescue！在一般的数据结构和计算方法的书藉中都有。

对样例数据 1，经过 Floyd 算法计算之后，货币之间的兑换汇率如下面左表所示。

对样例数据 2，经过 Floyd 算法计算之后，货币之间的兑换汇率如下面右表所示。

	USDollar	BritishPound	FrenchFranc
USDollar	1.05	0.525	5.25
BritishPound	2.10	1.05	10.50
FrenchFranc	0.2205	0.11025	1.1025

	USDollar	BritishPound	FrenchFranc
USDollar	0.995	0.50	5.00
BritishPound	1.99	0.995	10.00
FrenchFranc	0.19	0.095	0.95

编程时，对汇率 rate 取对数： $\log(\text{rate})$ ，这样在 Floyd 算法中，将汇率之间的相乘变成相

加，计算速度稍微快一点。套汇能够成功，就是货币兑换一圈之后回到自己时，汇率大于 1 ( $\log(\text{rate}) > 0$ )。所以从上面的计算结果看出，样例 1 的所有货币兑换之后，其汇率都大于 1，套汇是成功的；而样例 2 中，没有一个货币兑换之后的汇率大于 1，所以套汇是失败的。

### 【程序代码】

---

程序名称:	zju1092.c
题    目:	Arbitrage
提交语言:	C
运行时间:	00:00.06
运行内存:	420K

---

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#define zero 0.000001

int main()
{
    char name[30][300];           //存储所有的货币名称
    char from[300], to[300];      //一个货币名称
    double g[30][30];             //邻接矩阵
    double rate;                   //汇率
    int flag;                       //套汇成功标志
    int i,j,k;
    int n,m;                        //货币的数量，货币直接兑换的关系数
    int cases = 1;                  //测试例数
    while(scanf("%d", &n) && n)
    {
        memset(g, 0, sizeof(g));
        //读取所有的货币名称
        for(i=0; i<n; i++)
            scanf("%s", name[i]);
        //构建邻接矩阵
        scanf("%d", &m);
        for(i=0; i<m; i++)
        {
            scanf("%s%lf%s", from, &rate, to);
            //查找货币字符串所对应的序号
            for(j=0; j<30; j++)
                if (strcmp(from, name[j])==0) break;
            for(k=0; k<30; k++)
                if (strcmp(to, name[k])==0) break;
            g[j][k] = log(rate);      //取对数
        }
        //实现 Floyd 算法
```

```

        for(k=0; k<n; k++)
            for(i=0; i<n; i++)
                for(j=0; j<n; j++)
                {
                    rate = g[i][k]+g[k][j];
                    if(g[i][j]<rate) g[i][j] = rate;
                }
//判断套汇是否成功 (log(1)=0)
flag = 0;
for(i=0; i<n; i++)
    if (g[i][i]>zero) flag = 1;
printf("Case %d: ", cases++);
printf("%s\n", flag ? "Yes" : "No");
}
return 0;
}

```

## ZJU1117-Entropy<sup>[1、2]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768 KB

---

### Background

An entropy encoder is a data encoding method that achieves lossless data compression by encoding a message with "wasted" or "extra" information removed. In other words, entropy encoding removes information that was not necessary in the first place to accurately encode the message. A high degree of entropy implies a message with a great deal of wasted information; English text encoded in ASCII is an example of a message type that has very high entropy. Already compressed messages, such as JPEG graphics or ZIP archives, have very little entropy and do not benefit from further attempts at entropy encoding.

English text encoded in ASCII has a high degree of entropy because all characters are encoded using the same number of bits, eight. It is a known fact that the letters E, L, N, R, S and T occur at a considerably higher frequency than do most other letters in English text. If a way could be found to encode just these letters with four bits, then the new encoding would be smaller, would contain all the original information, and would have less entropy. ASCII uses a fixed number of bits for a reason, however: it's easy, since one is always dealing with a fixed number of bits to represent each possible glyph or character. How would an encoding scheme that used four bits for the above letters be able to distinguish between the four-bit codes and eight-bit codes? This seemingly difficult problem is

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1117>

[2] <http://www.acmgnyr.org/year2000/>

solved using what is known as a "prefix-free variable-length " encoding.

In such an encoding, any number of bits can be used to represent any glyph, and glyphs not presented in the message are simply not encoded. However, in order to be able to recover the information, no bit pattern that encodes a glyph is allowed to be the prefix of any other encoding bit pattern. This allows the encoded bitstream to be read bit by bit, and whenever a set of bits is encountered that represents a glyph, that glyph can be decoded. If the prefix-free constraint was not enforced, then such a decoding would be impossible.

Consider the text "AAAAABCD". Using ASCII, encoding this would require 64 bits. If, instead, we encode "A" with the bit pattern "00", "B" with "01", "C" with "10", and "D" with "11" then we can encode this text in only 16 bits; the resulting bit pattern would be "0000000000011011". This is still a fixed-length encoding, however; we're using two bits per glyph instead of eight. Since the glyph "A" occurs with greater frequency, could we do better by encoding it with fewer bits? In fact we can, but in order to maintain a prefix-free encoding, some of the other bit patterns will become longer than two bits. An optimal encoding is to encode "A" with "0" , "B" with "10", "C" with "110", and "D" with "111". (This is clearly not the only optimal encoding, as it is obvious that the encodings for B, C and D could be interchanged freely for any given encoding without increasing the size of the final encoded message.) Using this encoding, the message encodes in only 13 bits to "0000010110111 " , a compression ratio of 4.9 to 1 (that is, each bit in the final encoded message represents as much information as did 4.9 bits in the original encoding). Read through this bit pattern from left to right and you'll see that the prefix-free encoding makes it simple to decode this into the original text even though the codes have varying bit lengths.

As a second example, consider the text "THE CAT IN THE HAT". In this text, the letter "T" and the space character both occur with the highest frequency, so they will clearly have the shortest encoding bit patterns in an optimal encoding. The letters "C", "I" and "N" only occur once, however, so they will have the longest codes.

There are many possible sets of prefix-free variable-length bit patterns that would yield the optimal encoding, that is, that would allow the text to be encoded in the fewest number of bits. One such optimal encoding is to encode spaces with "00", "A" with "100", "C" with "1110", "E" with "1111", "H" with "110", "I" with "1010", "N" with "1011" and "T" with "01". The optimal encoding therefore requires only 51 bits compared to the 144 that would be necessary to encode the message with 8-bit ASCII encoding, a compression ratio of 2.8 to 1.

## Input

The input file will contain a list of text strings, one per line. The text strings will consist only of uppercase alphanumeric characters and underscores (which are used in place of spaces). The end of the input will be signaled by a line containing only the word "END" as the text string. This line should not be processed.

## Output

For each text string in the input, output the length in bits of the 8-bit ASCII encoding, the length

in bits of an optimal prefix-free variable-length encoding, and the compression ratio accurate to one decimal point.

## Example

### Input

```
AAAAABCD
THE_CAT_IN_THE_HAT
END
```

### Output

```
64 13 4.9
144 51 2.8
```

## Problem Source

Greater New York 2000

### 【题目大意】

通过剔除“无用”的信息，一致性编码（Entropy Encoder）方法能够实现数据的无损压缩。换句话说，一致性编码将忽略对精确编码本来就无关的信息。高度一致性意味着消息中有很多是无用的数据，用 ASCII 编码的英语文本就是一种高度一致性消息的例子。压缩的消息文档，如 JPEG 图像、ZIP 文档，一致性较低，对其进行一致性编码没有什么好处。

用 ASCII 编码的英语文本之所以具有高度一致性，原因在于编码长度相同：8 位。众所周知，字母 E, L, N, R, S 和 T 在英文中出现频率较高。如果能找到一种方法对这些字母用 4 位编码，仍然能表示原有信息，新的编码更短且具有较低的一致性。ASCII 编码使用固定长度编码的一个原因是使用方便，人们总是使用固定的位数表示字符。那么使用 4 位编码方案怎么区别是 4 位码还是 8 位码？这个难题用可变长无前缀码（“prefix-free variable-length”）来解决。

这种编码方法可以采用任意的位数来代表某个字符，当然消息里没有的字符不用编码。然而，为了能够还原信息，任一字符的代码都不是其他字符代码的前缀。这样就可以逐位读取编码位串，一组位串表示一个字符，即通过解码得到字符。如果没有无前缀（prefix-free）的限制，这种编码方案就行不通了。

考虑本文“AAAAABCD”。使用 ASCII 编码需要 64 位。如果使用“00”表示“A”，“01”表示“B”，“10”表示“C”，“1”表示“D”，那么只需 16 位就能编码这个文本，产生的位串是“0000000000011011”。然而这仍然是一个固定长度编码，但我们正使用 2 位表示一个字符而不是 8 位。因为“A”出现的频率较高，能否以比较少的位数获得更好的编码方法呢？事实上，这是可以做到的。但是为了达到无前缀编码，一些字符的位数将会比 2 位长。一种最优编码为“A”-“0”，“B”-“10”，“C”-“110”，“D”-“111”。（这不是唯一的最优编码方式。很明显，如果不增加编码信息的长度，B, C 和 D 的编码是可以互换的。）使用这种方法，消息编码的长度只有 13 位“000010110111”，压缩比 4.9:1（这意味着，编码后的消息每一位表示的信息相当于原先的 4.9 位）。尽管编码的长度是可变的，但从左至右逐位读取位串时，你会发现无前缀编码方法使解码变得简单。



第二个例子，考虑文本“THE CAT IN THE HAT”。字母“T”和空格符出现的频率最高，显然在最优编码中，这两个字符的编码位数应该最少。而字母“C”，“I”和“N”只出现一次，其编码长度将是最长的。

采用可变长无前缀码，有很多可能的编码方案产生最优编码，即以最少的位数对文本进行编码。一种最优编码是：空格-“00”，“A”-“100”，“C”-“1110”，“E”-“1111”，“H”-“110”，“I”-“1010”，“N”-“1011”和“T”-“01”。因此最优编码只要51位，而8位的ASCII编码需要144位，压缩比2.8:1。

### 输入格式

输入一连串的文本字符串，每行一个字符串。字符串只有大写字母、数字和下划线(代替空格)。如果一行只有一个“END”字符串，结束输入，这行也不必处理。

### 输出格式

对于输入的文本字符串，输出用8位ASCII码编码的长度，采用可变长无前缀码的最优编码的长度，压缩比（精确到小数点1位）。

## 【算法分析】

采用可变长无前缀码对文本进行压缩时，可以得到比较高的压缩比。本题要求计算文本的8位ASCII码编码长度，可变长无前缀码最优编码的长度和压缩比。

实现可变长无前缀码常用的方法是 Huffman 编码，关于这方面的知识，在离散数学、数据结构和算法分析与设计的书籍中有详细的介绍，这里只介绍与本题有关的算法实现。

### （1）构造权重数组

实现 Huffman 编码的第一步是计算每个字符出现的频率，以数组 freq 存放频率：

```
int freq[64];
```

遍历文本 text，统计每个字符出现的次数即可。这里使用了坐标映射，“A”~“Z”映射到0~25，“\_”映射为26。

### （2）构造 Huffman 树

使用数组 tree 存放 Huffman 树：

```
struct {
    int lchild, rchild, parent;
}tree[64];
```

霍夫曼提出贪心算法构造可变长无前缀码，因此该编码方案称为霍夫曼编码。该算法以自底向上的方式构造表示可变长无前缀码的二叉树  $T$ ，这是一棵完全二叉树。

算法以  $|C|$  个叶结点开始，执行  $|C|-1$  次的“合并”运算后产生最终所要求的树  $T$ 。文本字符集中每一字符  $c$  出现的频率是  $\text{freq}[c]$ 。在  $\text{freq}$  数组中查找两个具有最小频率的树，而且不在森林中，就将这两棵树合并，产生一棵新的树，其频率为合并的2棵树的频率之和，将新树并入森林中。经过  $n-1$  次的合并后，森林中只剩下一棵树，即所要求的树  $T$ 。

如果采用 C++ 标准模板库函数 `priority_queue()`（头文件 `#include <queue>`），编程就轻松很多。

### （3）计算编码长度

文本的8位ASCII码编码长度，就是字符串的长度。而文本的可变长无前缀码的最优编码长度，需要通过频率数组  $\text{freq}$  计算。

如果文本中只有一种字符，则每个字符只需1位编码，其长度就是字符的个数。否则，从二叉树的根开始遍历，搜索每个叶结点的码长，用码长乘以该叶结点所表示字符出现的频率，

累计求和即可。

(4) 样例分析

对于样例 1，字符权重数组 `freq` 的值如下：

下标	0	1	2	3
权重	5	1	1	1
字母	A	B	C	D

构造的 Huffman 树如图 8-3 所示：

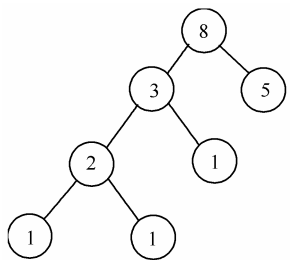


图 8-3 样例 1 的 Huffman 树

从图 8-3 看出，A 的码长为 1，D 的码长为 2，B 和 C 的码长为 3，则可变长无前缀码的最优编码长度为码长与相应字符权重的乘积和为

$$1 \times 5 + 3 \times 1 + 3 \times 1 + 2 \times 1 = 13$$

【程序代码】

程序名称	zju1117.c
题    目：	Entropy
提交语言：	C
运行时间：	0ms
运行内存：	160KB

```
#include <stdio.h>
#include <string.h>

//存放 Huffman 树
struct {
    int lchild, rchild, parent;
}tree[64];

char text[100];
int freq[64];
int optimal;

//文本
//每个字符出现的频率
//可变长无前缀码的最优编码长度

//递归计算可变长无前缀码的最优编码长度
//形参 n 是结点编号，d 是码长
```

```

void entropy(int n, int d)
{
    if(freq[n]==0) return;
    if(n<31) optimal += freq[n]*d;           //累加该字符的编码长度
    else
    {
        entropy(tree[n].lchild, ++d);       //左子树，码长加 1
        entropy(tree[n].rchild, d);         //右子树，码长与左子树相同
    }
}

int main()
{
    int i,j;
    int left,right;
    while(gets(text))
    {
        if (strcmp(text,"END")==0) break;   //结束标志
        int length = strlen(text);          //字符串长度
        memset(tree, 255, sizeof(tree));
        memset(freq, 0, sizeof(freq));
        //构造权重数组
        for(i = 0; i<length; i++)
            if (text[i]!='_') freq[26]++;
            else freq[text[i]-'A']++;
        int node = 30;
        while(1)
        {
            //查找第一个权重最小的结点（树）
            int min = 1000;                  //最小权重
            for(j = 0; j<=node; j++)
                if(tree[j].parent== -1)      //此树不在树林中
                    if(min>freq[j] && freq[j])
                    {
                        min = freq[j];
                        left = j;             //作为左子树
                    }
            //查找第二个权重最小的结点（树）
            min = 1000;
            for(j = 0; j<=node; j++)
                if(tree[j].parent== -1)
                    if(j!=left && min>freq[j] && freq[j])
                    {

```

```

        min = freq[j];
        right = j;                //作为右子树
    }
    if (min==1000) break;         //已经是森林
    //合并两个最小权重树的权重
    freq[++node] = freq[left] + freq[right];
    //构造 Huffman 树
    tree[node].lchild = left;
    tree[node].rchild = right;
    tree[node].parent = -1;
    tree[left].parent = node;
    tree[right].parent = node;
}
optimal = 0;
if(node==30) optimal = length;   //只有一个字符
else entropy(node,0);           //从根结点开始计算,其码长为 0
length *= 8;                    //ASCII 码编码长度
printf("%d %d %.1lf\n", length, optimal, 1.0*length/optimal);
}
return 0;
}

```

## ZJU1118-N-Credible Mazes<sup>[1、2、3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

### Background

An n-tersection is defined as a location in n-dimensional space, n being a positive integer, having all non-negative integer coordinates. For example, the location (1,2,3) represents an n-tersection in three dimensional space. Two n-tersections are said to be adjacent if they have the same number of dimensions and their coordinates differ by exactly 1 in a single dimension only. For example, (1,2,3) is adjacent to (0,2,3) and (2,2,3) and (1,2,4), but not to (2,3,3) or (3,2,3) or (1,2). An n-teresting space is defined as a collection of paths between adjacent n-tersections.

Finally, an n-credible maze is defined as an n-teresting space combined with two specific n-tersections in that space, one of which is identified as the starting n-tersection and the other as the ending n-tersection.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1118>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1522>

[3] <http://www.acmgnyr.org/year2000/>

## Input

The input file will consist of the descriptions of one or more  $n$ -credible mazes. The first line of the description will specify  $n$ , the dimension of the  $n$ -teresting space. (For this problem,  $n$  will not exceed 10, and all coordinate values will be less than 10.) The next line will contain  $2n$  non-negative integers, the first  $n$  of which describe the starting  $n$ -tersection, least dimension first, and the next  $n$  of which describe the ending  $n$ -tersection. Next will be a nonnegative number of lines containing  $2n$  non-negative integers each, identifying paths between adjacent  $n$ -tersections in the  $n$ -teresting space. The list is terminated by a line containing only the value  $-1$ . Several such maze descriptions may be present in the file. The end of the input is signalled by space dimension of zero. No further data will follow this terminating zero.

## Output

For each maze output it's position in the input; e.g. the first maze is "Maze #1", the second is "Maze #2", etc. If it is possible to travel through the  $n$ -credible maze's  $n$ -teresting space from the starting  $n$ -tersection to the ending  $n$ -tersection, also output "can be travelled" on the same line. If such travel is not possible, output "cannot be travelled" instead.

Example

### Input

```
2
0 0 2 2
0 0 0 1
0 1 0 2
0 2 1 2
1 2 2 2
-1
3
1 1 1 1 2 3
1 1 2 1 1 3
1 1 3 1 2 3
1 1 1 1 1 0
1 1 0 1 0 0
1 0 0 0 0 0
-1
0
```

### Output

```
Maze #1 can be travelled
Maze #2 cannot be travelled
```

## Problem Source

Greater New York 2000

## 【题目大意】

**n-tersection** 定义为  $n$  维空间中的一个坐标位置,  $n$  是正整数, 坐标点也都是正整数。例如, 坐标 (1, 2, 3) 就是三维空间中的一个 **n-tersection**。如果空间中两点的维数相同且在某一个方向上坐标数只相差 1, 那么这两个 **n-tersections** 就是相邻的。例如, (1, 2, 3) 与 (0, 2, 3)、(2, 2, 3) 和 (1, 2, 4) 是相邻的, 但与 (2, 3, 3)、(3, 2, 3) 和 (1, 2) 就不是相邻的。**n-teresting** 空间定义为相邻 **n-tersection** 所构成路径的集合。

最后, **n-credible** 迷宫定义为两个指定的 **n-teresting** 构成的一个 **n-teresting** 空间, 其中一个作为起始 **n-tersection**, 而另一个作为结尾 **n-tersection**。

### 输入格式

输入一个或多个 **n-credible** 迷宫。对每个迷宫, 第一行是  $n$ , **n-teresting** 空间的维度 ( $n$  不超过 10, 坐标点的数值小于 10)。第二行是  $2n$  个非负整数, 头  $n$  个数表示起始 **n-tersection**, 后  $n$  个数是结尾 **n-tersection**, 维数低的在先。接下来有多行, 每行  $2n$  个非负整数, 描述 **n-teresting** 空间里相邻 **n-tersection** 构成的路径。遇到一行只有 -1 时结束。输入文件中可能有多个这样的迷宫。遇到 0 维空间时, 输入结束。

### 输出格式

对于每个迷宫, 输出它在输入中的编号。例如, 第一个迷宫是 “Maze #1”, 第二个是 “Maze #2”, 等等。如果 **n-credible** 迷宫的 **n-teresting** 空间里, 从起始 **n-tersection** 能够到达结尾 **n-tersection**, 输出 “can be travelled”, 否则输出 “cannot be travelled”。

## 【算法分析】

在  $n$  维空间中的两个点, 如果在某一个方向上坐标数只相差 1, 就称为是相邻点, 一系列相邻点构成  $n$  维空间的一条路径。给定起点和终点, 再给出多组相邻点, 通过已知相邻点构成的路径, 能否从起点到达终点。

给出的相邻点肯定是合法的。如果真当  $n$  维空间解题的话, 编码肯定是很复杂的。但是把  $n$  维空间映射成一维空间的话, 解题就容易了。一种方法是通过 Hash 表映射, 编码也是比较复杂。已知坐标点的值小于 10, 也就是个位数, 那就可以把  $n$  维空间的值并在一起, 构成一个字符串。处理字符串就像成语接龙, 一个字符串的后一半等于另一个字符串的前一半, 就接下去。更进一步简化, 把相邻的两个字符串映射成数字, 就可以使用并查集算法, 编码就变得更简单。

### (1) 构造字符串

对  $n$  维空间的每一个坐标, 读出以后加 ‘0’, 并入字符串中。

### (2) 实现字符串映射成数字, 构造并查集

该功能由函数实现:

```
int build(char path[], int &count);
```

形参 **path** 是待映射的字符串, **count** 是已经使用的字符编号, 返回值是字符串 **path** 映射的编号。为了方便字符串与数字的映射, 这里使用了 C++ 标准模板库函数 **map()** 容器。其实建立字符串数组, 映射也是方便的。

映射结果存放在数组中:

```
map<string,int> maps;
```

样例 2 的坐标映射如表下:

key	000	100	110	111	112	113	123
value	7	6	5	4	1	2	3

从数组 `maps` 看出，编号是以坐标出现的先后顺序进行的。

### (3) 并查集搜索

该功能由函数实现：

```
int findhead(int number);
```

返回值是形参 `number` 的根。通过递归的方法查找 `number` 的根。

样例 2 的并查集如表下：

下标	0	1	2	3	4	5	6	7
值	0	2	3	3	5	6	7	7

起点坐标是 (1, 1, 1)，映射的数字是 4；结尾坐标是 (1, 2, 3)，映射的数字是 3。搜索并查集，得到 3 的根是 3，而 4 的根是 7；所以这两点之间是不可达的。

## 【程序代码】

程序名称： zju1118.cpp  
 题 目： N-Credible Mazes  
 提交语言： C++  
 运行时间： 0ms  
 运行内存： 184KB

```
#include<iostream>
#include<map>
using namespace std;

map<string,int> maps; //实现字符串映射的 map() 容器
int set[200];         //存放并查集

//搜索并查集，查找 number 的根
int findhead(int number)
{
    if(set[number]==number) return number;
    else
    {
        set[number] = findhead(set[number]);
        return set[number];
    }
}

//实现字符串映射成数字，构造并查集，返回值是字符串 path 的编号
//形参 path 是待映射的字符串，count 是已经使用的字符编号
int build(char path[], int &count)
{

```

```

int index;
//新的字符串
if(maps.find(string(path))==maps.end())
{
    maps[string(path)] = ++count;           //映射
    index = count;
    set[count] = count;                     //并查集的一个根
}
//已经映射的字符串, 直接返回编号
else index = maps[string(path)];
return index;
}

int main()
{
    int n;                                //n 维空间
    int number = 1;
    while(scanf("%d",&n) && n)
    {
        int i;
        //构造起点坐标的字符串
        int p;
        char starting[15], ending[15];
        for(i = 0; i<n; i++)
        {
            scanf("%d", &p);
            starting[i] = p+'0';
        }
        starting[n] = 0;
        //构造结尾坐标的字符串
        for(i = 0; i<n; i++)
        {
            scanf("%d", &p);
            ending[i] = p+'0';
        }
        ending[n] = 0;
        //路径处理
        int count = 0;                    //映射的数字编号
        maps.clear();
        char path[15];                    //一条相邻路径
        while(scanf("%d", &p))
        {
            if(p== -1) break;

```



```

    int start, end;
    //构造相邻路径起点坐标的字符串
    path[0] = p+'0';
    for(i = 1; i<n; i++)
    {
        scanf("%d", &p);
        path[i] = p+'0';
    }
    path[n] = 0;
    //实现字符串映射成数字, 构造并查集
    start = build(path, count);
    //构造相邻路径结尾坐标的字符串
    for(i = 0; i<n; i++)
    {
        scanf("%d", &p);
        path[i] = p+'0';
    }
    path[n] = 0;
    //实现字符串映射成数字, 构造并查集
    end = build(path, count);
    int h1 = findhead(start);
    int h2 = findhead(end);
    if(h1!=h2) set[h1] = h2; //一个新的根结点
}
int possible = 1;
//判断起点坐标和结尾坐标是否在路径中
if(maps.find(string(starting))==maps.end()
    || maps.find(string(ending))==maps.end())
    possible = 0; //不在路径中
else
{
    //判断起点坐标和结尾坐标的根结点是否相同
    int start = maps[string(starting)];
    int end = maps[string(ending)];
    int h1 = findhead(start);
    int h2 = findhead(end);
    if(h1!=h2) possible = 0; //根结点不相同
}
if (possible) printf("Maze #%d can be travelled\n",number++);
else printf("Maze #%d cannot be travelled\n",number++);
}
return 0;
}

```

Time Limit: 1 Second

Memory Limit: 32768KB

## Background

Consider the two networks shown below. Assuming that data moves around these networks only between directly connected nodes on a peer-to-peer basis, a failure of a single node, 3, in the network on the left would prevent some of the still available nodes from communicating with each other. Nodes 1 and 2 could still communicate with each other as could nodes 4 and 5, but communication between any other pairs of nodes would no longer be possible.

Node 3 is therefore a Single Point of Failure (SPF) for this network. Strictly, an SPF will be defined as any node that, if unavailable, would prevent at least one pair of available nodes from being able to communicate on what was previously a fully connected network. Note that the network on the right has no such node; there is no SPF in the network. At least two machines must fail before there are any pairs of available nodes which cannot communicate.

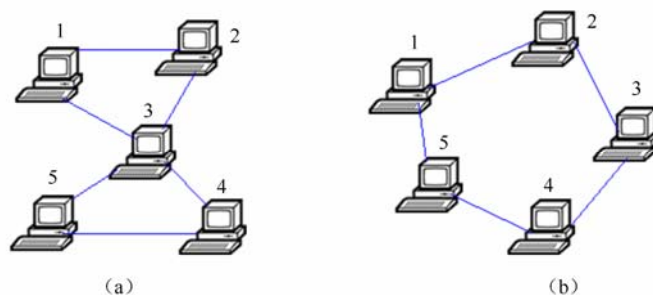


Figure 8-4

## Input

The input will contain the description of several networks. A network description will consist of pairs of integers, one pair per line, that identify connected nodes. Ordering of the pairs is irrelevant; 1 2 and 2 1 specify the same connection. All node numbers will range from 1 to 1000. A line containing a single zero ends the list of connected nodes. An empty network description flags the end of the input. Blank lines in the input file should be ignored.

## Output

For each network in the input, you will output its number in the file, followed by a list of any SPF nodes that exist.

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1119>

[2] <http://www.acmgnyr.org/year2000/>

The first network in the file should be identified as " Network #1 ", the second as " Network #2 ", etc. For each SPF node, output a line, formatted as shown in the examples below that identify the node and the number of fully connected subnets that remain when that node fails. If the network has no SPF nodes, simply output the text " No SPF nodes " instead of a list of SPF nodes.

### Example

#### Input

1 2	1 2	1 2	0
5 4	2 3	2 3	
3 1	3 4	3 4	
3 2	4 5	4 6	
3 4	5 1	6 3	
3 5	0	2 5	
0		5 1	
		0	

#### Output

```
Network #1
  SPF node 3 leaves 2 subnets
Network #2
  No SPF nodes
Network #3
  SPF node 2 leaves 2 subnets
  SPF node 3 leaves 2 subnets
```

### Problem Source

Greater New York 2000

#### 【 题目大意 】

观察下面两个网络。在基于点对点（peer-to-peer）的网络中，假定数据只能在结点直接相连的网络中传输，那么图 8-4（a）网络中的一个结点（如 3 号结点）故障时，就阻断了仍然相连的结点之间的通信。结点 1、2 之间和结点 4、5 之间仍然畅通，但是这两部分结点之间的联系中断了。

因此结点 3 称为该网络的单点故障（Single Point of Failure，SPF）。严格来讲，SPF 定义为如果某个结点失效的话，将会使原来完全相通的网络，至少被阻断成了两部分网络。注意：图 8-4（b）没有 SPF 结点。任意一对结点通信中断，至少使得两台机器失效。

#### 输入格式

输入有多组网络。对每个网络，有多对整数，每行一对，表示连通的结点。整数对是无序的；1 2 和 2 1 是指同一个连接。结点编号范围是 1~1000。一行只有一个 0 时，该网络的数据就结束了。遇到空网络时结束输入。输入时的空行应予忽略。

## 输出格式

对每个网络，输出网络编号及存在的 SPF 结点列表。

第一个网络定义为“Network #1”，第二个网络定义为“Network #2”，等等。对每个 SPF 结点，输出一行，格式如输出样例所示，指出 SPF 结点，以及该结点失效时产生的完全连通的子网数。如果网络中无 SPF 结点，只需输出“No SPF nodes”。

## 【算法分析】

一个完全连通的网络，如果某个结点故障，将网络分成了至少两个子网络，则该结点故障就称为单点故障（Single Point of Failure, SPF）。这正是图的连通性问题，见参考文献[8]，严尉敏的数据结构教材，“关结点和重连通分量”一节有详细的算法介绍。因此这里只根据题目的需要作简单的分析。关结点也称为割点。

设  $G=(V, E)$  是一个连通无向图，若删除顶点  $v$  以及  $v$  相关的边后，将图的一个连通分量分割为两个或两个以上的连通分量，则称顶点  $v$  为该图的一个关结点。一个没有关结点的连通图称为重连通图。

在重连通图中，任意一对顶点之间至少存在两条路径，则在删去某个顶点以及依附于该顶点的各边时也不破坏图的连通性。若在连通图上至少删去  $k$  个顶点才能破坏图的连通性，则称  $k$  为此图的连通度。

利用深度优先搜索算法可以求图的关结点，并由此可判别图是否重连通。

从任一点出发深度优先遍历得到优先生成树，对树中任一顶点  $V$  而言，其儿子结点为邻接点。由深度优先生成树可得出两类关结点的特性：

① 若生成树的根有两棵或两棵以上的子树，则此根顶点必为关结点。因为图中不存在连接不同子树顶点的边，若删除此结点，则树便成为森林。

② 若生成树中某个非叶子顶点  $v$ ，其某棵子树的根和子树中的其他结点均没有指向  $v$  的祖先的回边，则  $v$  为关结点。因为若删除  $v$ ，则其子树和图的其他部分分割开来。

### （1）数据结构

无向图用邻接矩阵表示：

```
bool G[1000][1000];
```

由于是无向图，所以数组  $G$  是关于对角线对称的。

结点的最大编号：

```
int n;
```

### （2）深度优先搜索，遍历结点 $v$ 的所有邻接点

由函数 DFS 实现：

```
void DFS(int v, int d)
```

访问结点  $v$  的顺序号为  $d$ ，存放在数组  $order$  中。

通过 DFS 遍历结点  $v$  可以达到的最小顺序号，存放在数组  $low$  中：

```
int low[1000];
```

如果结点  $v$  没有遍历过，则遍历结点  $v$  的所有邻接点  $i$  ( $1 \leq i \leq n$ )。当  $(v, i)$  有边时，如果结点  $i$  已经访问过，则  $low[v] = \min(low[v], order[i])$ ，否则计算结点  $i$  的最小顺序号  $low[i]$ ，则  $low[v] = \min(low[v], low[i])$ 。

### （3）计算 SPF

由函数 SPF 实现：

```
void SPF();
```

对每个结点  $i$  ( $1 \leq i \leq n$ ), 判断其是否是关结点。如果是的话, 它又把连通分量分割成几个子连通分量。子网计数: 对所有的邻接点  $j$  ( $1 \leq j \leq n$ ), 如果有  $\text{father}[j]=i$  并且 ( $\text{low}[j] \geq \text{order}[i]$ ), 这就是一个子网。如果结点  $j$  是一个内部结点, 考虑到父结点以上的子网, 则子网数还要加 1。

## 【程序代码】

---

```
程序名称:      SPF
题    目:      zju1119.cpp
提交语言:      C++
运行时间:      0ms
运行内存:      1168KB
```

---

```
#include<stdio.h>
#include<memory.h>

const int Node = 1000;
int n;                                     //结点的最大编号
bool G[Node][Node];                       //无向图的邻接矩阵
int father[Node];                         //DFS 遍历的前趋
int order[Node];                          //DFS 遍历的序号号
int low[Node];                            //DFS 遍历的结点可以达到的最小序号号
bool visited[Node];                       //DFS 遍历的标志

//深度优先搜索, 遍历结点 v 的所有邻接点, 序号号为 d
void DFS(int v, int d)
{
    if (visited[v]) return;
    low[v] = d;
    order[v] = d;
    visited[v] = true;                    //置访问标志
    //遍历结点 v 的所有邻接点, 构建数组 low 和 father
    for (int i = 0; i < n; i++) {
        if (G[v][i]) {                   //如果有边
            //结点 i 已经遍历过, <v, i>是后向边
            if (visited[i])
                low[v] = low[v]<order[i]?low[v]:order[i];
            else {
                //结点 i 是第一次访问, <v, i>是前向边
                father[i] = v;             //结点 v 是结点 i 的父结点
                DFS(i, d+1);
                low[v] = low[v]<low[i]?low[v]:low[i];
            }
        }
    }
}
```

```

}

//计算 SPF
void SPF()
{
    //初始化数组
    for (int i = 0; i < n; i++)
    {
        visited[i] = false;
        father[i] = -1;
        low[i] = 0;
        order[i] = 0;
    }
    //找出所有的连通分量
    for (int i = 0; i < n; i++)
        if (!visited[i]) DFS(i, 1);
    //对每个结点, 判断其是否是关结点; 计算子网数并输出结果
    bool find = false;
    for (int i = 0; i < n; i++)
    {
        //如果结点 i 是关结点, 计算它把连通分量分成了几个子连通分量
        int subnet = 0; //子网计数
        for (int j = 0; j < n; j++)
            //对 i 的所有儿子结点
            if ((father[j] == i) && (low[j] >= order[i]))
                subnet++;
        if (subnet > 0) { //有 SPF
            if (father[i] == -1) { //i 是根结点
                if (subnet > 1) {
                    find = true;
                    printf(" SPF node %d leaves %d subnets\n", i+1, subnet);
                }
            }
            else { //i 是内部结点
                find = true;
                printf(" SPF node %d leaves %d subnets\n", i+1, subnet+1);
            }
        }
    }
    if (!find) {
        printf(" No SPF nodes\n");
    }
}

int main(){

```

```

int i,j;
int number = 0;
memset(G, 0, sizeof(G));
//读取数据, 构造邻接矩阵
while(scanf("%d", &i)) {
    if(i==0) { //一个无向图输入结束
        if(number) putchar('\n');
        printf ("Network #%d\n", ++number);
        SPF();
        scanf("%d", &i);
        if (i==0) break; //输入文件结束
        memset(G, 0, sizeof(G));
    }
    n = -1;
    scanf("%d", &j);
    if(i>n) n = i; //计算最大的结点编号
    if(j>n) n = j;
    G[i-1][j-1] = G[j-1][i-1]=1; //无向图是对称的
}
}

```

## ZJU1127-Roman Forts<sup>[1、2]</sup>

Time Limit: 1 Second

Memory Limit: 32768KB

This program will acquaint you with the judicious method used by rulers of the ancient Roman empire to choose the locations of their military forts.

You are given a collection of cities.

The cities will be labeled by upper case letters in alphabetical order, starting with 'A'.

Given any pair of cities, there is one and only one path between them (see Figure 8-5).

This path may be a direct connection (as between J and C in Figure 8-5) or it may pass through a number of intermediate cities (as between J and B in Figure 1).

In the example of Figure 1, the path from J to B passes through C, G, and F.

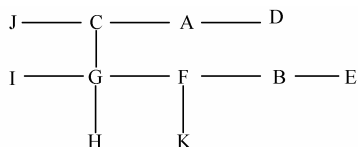


Figure 8-5

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1127>

[2] <http://acm.uva.es/archive/nuevportal/data/problem.php?p=2082>

The actual number of miles between cities is irrelevant. The distance between a pair of cities is defined as the number of hops (each "hop" being taken over a over a direct connection) between them.

In Figure 8-5, the number of hops between J and C is 1, and the number of hops between J and B is 4.

Initially, only one of the cities is fortified (by a military fort). As your program executes, additional cities will be fortified.

If a city is not fortified, we say it is vulnerable.

Furthermore, if a city is vulnerable, its degree of vulnerability is defined as the shortest distance between it and a fortified city.

When one or more cities have already been fortified, the location of an additional fort is chosen by a simple rule: it is the most vulnerable city (the one having the highest degree of vulnerability).

In case of a tie, choose the alphabetically first city among those tied as the most vulnerable city.

## Sample Input

```
11 J 5
A D
B E
F B
C G
F G
C A
G H
G I
J C
K F
```

The first line of the input contains three items of information, separated by one or more blank spaces:

- The total number of cities in the given collection. It will be in the range 3...26. In this example, it is 11, which means that the cities are labeled A...K.

- An upper case letter in the valid range of cities, denoting the city which is initially fortified.

- The total number of cities that are to be fortified, including the initially fortified one.

It will be at least three and not greater than the total number of cities in the collection

Each of the remaining lines of the input file will contain two upper case letters, separated by one or more blank spaces.

Each letter will be in the range of labels of cities in the given collection.

Each pair of letters on one line of input determines a direct connection between two cities.

The direct connections listed in the input file shown on the right are identical to the direct connections of Figure 8-5.

The set of direct connections listed in the input file will satisfy all previously stated conditions.

All lines of input will be free of leading or trailing blank spaces.



The output will list the cities to be fortified, as upper case letters, in the correct order.

## Sample Output

Program 8 by team X

J E D H K

End of program 8 by team X

Explanation of the Output:

When there is only one fortified city, the degree of vulnerability of each vulnerable city is its distance from the fortified city.

Initially, with J being the only fortified city, E is the most vulnerable city with degree of vulnerability 5. With J and E being fortified, D, H, I, and K contend for being the most vulnerable city, all having degree of vulnerability 3.

For example, the distance from D to E is 6, the distance from D to J is 3; the smaller of these two numbers is 3. Therefore, the degree of vulnerability of D is 3.

The alphabetically first city among the four contenders (D, H, I, and K) is D, which will be the next city to be fortified.

Similarly, H, and then K, will be selected as the next city to be fortified. At this point, the number of fortified cities is 5 (as specified in the input), and the program terminates

Note that city I is not fortified, because its degree of vulnerability drops from 3 to 2 once H is fortified, making K the most vulnerable city.

Formatting details:

The upper case letters denoting the cities in the output will be separated by one blank space.

There will not be any blank lines or leading blank spaces in the output.

## Problem Source

Rocky Mountain 2000

### 【题目大意】

这个问题将使你熟悉古罗马帝国选择军事堡垒地址的智慧。

输入一系列城市。

城市以大写字母命名，按字母表顺序，从“A”开始。

城市之间只有一条通道（见图 8-5）。

城市间可能是直接相连的（如图 8-5 中 J 和 C），或者城市之间还有一些城市（如图 8-5 中 J 和 B）。

图 8-5 中，从 J~B 的通道经过 C、G 和 F。

城市间的实际距离是不同的。城市之间的距离用跳数表示（一个直接连接称为一“跳”）。

图 8-5 中，J 和 C 之间的跳数是 1，而 J 和 B 之间的跳数是 4。

最初，只有一个城市构筑堡垒（军事堡垒）。随着程序的运行，将有更多的城市构筑堡垒。

如果一个城市没有堡垒，我们说它是脆弱的。

如果一个城市是脆弱的，那么它的脆弱程度就定义为它和拥有堡垒城市之间的最短距离。

当一个或多个城市已经拥有堡垒时，那么下一个堡垒的选取地址遵循一个简单的规则：防御最脆弱的城市（脆弱程度最高）。

如果剩下的城市脆弱程度都一样，那么选字母顺序排在最前的那座城市。

### 输入格式

输入的第一行包括三项，由一个或多个空格分隔：

- 城市的总数，范围是 3~26。本例中是 11，意思是城市名为 A~K。
- 一个大写字母，在城市的有效名称内。表示最初拥有堡垒的城市。
- 需要构筑堡垒的城市数量，包括最初已有堡垒的城市。最少三个城市，最多不会超过城市总数。

其余各行包括两个大写字母，之间有一个或多个空格。

所有的字母，都是有效的城市名称。

输入的每行一对字母，表示该对城市间是直接相连的。

输入的每个连接，与图 8-5 中的连接是一样的。

输入的每对城市都满足上述条件。

输入的每行之前和末尾都没有空格。

### 输出格式

输出需要构筑堡垒的城市，以大写字母的顺序。

当只有一个城市拥有堡垒时，那么其他城市的脆弱程度就是到该堡垒城市的距离。

初始时，只有城市 J 是有堡垒的，E 是脆弱程度最高的，跳数是 5。当 J 和 E 拥有堡垒后，D、H、I 和 K 的脆弱程度最高，都是 3。

例如，从 D~E 的距离是 6，从 D~J 的距离是 3；较小的数是 3。因此，D 的脆弱程度是 3。

D、H、I 和 K 4 个城市按字母顺序的第一个城市是 D，也就是下一个要构筑堡垒的城市。

同样，然后选择 H，K。此时，拥有堡垒的城市数是 5（输入中所示），程序结束。

注意：城市 I 没有构筑堡垒，因为当 H 筑好堡垒后，I 的脆弱程度从 3 降到 2，使得 K 成为脆弱程度最高的城市。

### 格式细节：

输出时，表示城市的大写字母由一个空格隔开。

没有多余的空行或者前导空格。

## 【算法分析】

有很多城市，城市之间的距离用跳数（hop）表示。当一个城市有堡垒时，其他没有堡垒的城市就是脆弱的，其脆弱程度就是该城市到有堡垒的城市之间的最短距离，然后选择脆弱程度最高的城市构筑堡垒。一直到构筑的堡垒数达到要求为止。显然算法的关键是选择脆弱程度最高的城市，这实际上是计算源点到其余各顶点的最短路径，常用 Dijkstra 算法。

### （1）数据结构

城市总数：

```
int city;
```

表示城市的邻接矩阵，城市之间直接连接时，值为 1，否则为 0：

```
char graph [26][26];
```

城市构筑堡垒的情况，如果已经构筑堡垒，值为 1，否则为 0：

```
char fortify [26];
```

需要构筑堡垒的数量，包含第一个已经构筑堡垒的城市：

```
int total;
```

第一个构筑堡垒的城市：

```
char first;
```

(2) 确定下一个要构筑堡垒的城市

这是最短路径算法。关于 Dijkstra 算法的理论知识，请参考相关的数据结构和算法分析书籍，特别推荐清华大学出版社出版的严蔚敏编写的教材。这里只讨论与本题相关的算法实现。

① 实现 Dijkstra 算法的数据结构

引进辅助向量  $S$ ，表示当前已经求得最短路径的城市集合：

```
int searched [26];
```

用数组保存城市的脆弱程度：

```
int hop [26];
```

② 数组 hop 初始化

对城市  $i$  ( $0 \leq i < \text{city}$ )，如果已经有堡垒，则  $\text{hop}[i]=0$ ，否则  $\text{hop}[i]=\infty$ 。

③ 查找具有最短路径的城市  $k$

对不在集合 searched 中的城市，查找城市  $k$ ，使得：

$\text{hop}[k]=\min\{\text{hop}[j]\}$ , ( $0 \leq j < \text{city}$ , 且  $\text{searched}[j]=0$ )

令  $\text{searched}[k]=1$ ，将城市  $k$  并入集合 searched 中。

④ 更新当前最短路径

对城市  $j$  ( $0 \leq j < \text{city}$ , 且  $\text{searched}[j]=0$ )，如果城市  $(k, j)$  有连接，且经过城市  $k$  会使距离减少，则调整距离：

$\text{hop}[j]=\text{hop}[k]+1$  ( $\text{graph}[k][j]=1$ , 且  $\text{hop}[k]+1 < \text{hop}[j]$ )

⑤ 重复操作③和④共  $\text{city}-1$  次，就计算出所有没有堡垒城市的最短路径

⑥ 查找跳数最多（即脆弱程度最高）的城市编号

在数组 hop 中查找最大值，其下标就是下一个要构筑堡垒的城市。

## 【程序代码】

---

程序名称：	zju1127.c
题    目：	Roman Forts
提交语言：	C
运行时间：	0ms
运行内存：	160KB

---

```
#include <stdio.h>
#include <memory.h>

int city; //城市总数
char graph [26][26]; //表示城市的邻接矩阵
char fortify [26]; //城市构筑堡垒的情况
```

```

//根据最短路径算法，确定下一个要构筑堡垒的城市
int Dijkstra()
{
    int i, j, k;
    int searched [26]; //已经求得最短路径的城市集合
    int hop [26]; //城市的脆弱程度
    for ( i = 0; i < city; i ++ )
        if ( fortify [i] ) hop [i] = 0; //有堡垒的城市
        else hop [i] = 100; //没有堡垒的城市,100 相当于 $\infty$ 
    //根据 Dijkstra 算法，计算所有没有堡垒城市的最短路径
    memset(searched, 0, sizeof(searched));
    for ( i = 0; i < city; i ++ ) {
        //查找具有最短路径的城市
        k = -1;
        for ( j = 0; j < city; j ++ )
            if ( !searched [j] )
                if ( k < 0 || hop [j] < hop [k] ) k = j;
        if ( k < 0 ) break; //每个城市都处理完毕
        //将城市 k 加入到已经求得最短路径的城市集合中
        searched [k] = 1;
        //更新当前最短路径
        for ( j = 0; j < city; j ++ )
            if ( !searched [j] )
                if ( graph [k][j] && hop [k] + 1 < hop [j] )
                    hop [j] = hop [k] + 1;
    }
    //查找跳数最多（即脆弱程度最高）的城市编号
    j = -1;
    for ( i = 0; i < city; i ++ )
        if ( !fortify [i] ) //如果该城市还没有堡垒
            if ( j < 0 || hop [j] < hop [i] ) j = i;
    fortify [j] = 1; //标记该城市有堡垒
    return j;
}

int main ()
{
    int i;
    int total; //需要构筑堡垒的数量
    char first, str[10]; //第一个构筑堡垒的城市
    char a , b;
    scanf("%d%s%d\n", &city, str, &total);
    first = str[0];
    memset (fortify, 0, sizeof(fortify));
    //构造邻接矩阵
    memset (graph, 0, sizeof(graph));

```

```

    for ( i = 1; i < city; i ++ )
    {
        scanf("%c %c\n", &a, &b);
        graph[a-'A'][b-'A'] = 1;
        graph[b-'A'][a-'A'] = 1;
    }
    fortify [first - 'A'] = 1; //标记第一个堡垒
    printf("Program 8 by team X\n");
    printf("%c", first); //输出第一个堡垒
    for (i = 1; i < total; i ++ )
        printf(" %c", Dijkstra() + 'A'); //输出其他的堡垒(total-1个)
    printf("\n");
    printf("End of program 8 by team X\n");
    return 0;
}

```

## ZJU1130-Ouroboros Snake<sup>[1、2、3]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768KB

---

Ouroboros is a mythical snake from ancient Egypt. It has its tail in its mouth and continuously devours itself.

The Ouroboros numbers are binary numbers of  $2^n$  bits that have the property of "generating" the whole set of numbers from 0 to  $2^n - 1$ . The generation works as follows: given an Ouroboros number, we place its  $2^n$  bits wrapped in a circle. Then, we can take  $2^n$  groups of  $n$  bits starting each time with the next bit in the circle. Such circles are called Ouroboros circles for the number  $n$ . We will work only with the smallest

Ouroboros number for each  $n$ .

Example: for  $n = 2$ , there are only four Ouroboros numbers. These are 0011;0110;1100; and 1001. In this case (Figure 8-6), the smallest one is 0011. Here is the Ouroboros circle for 0011:

The table describes the function  $o(n; k)$  which calculates the  $k$ -th number in the Ouroboros circle of the smallest Ouroboros number of size  $n$ . This function is what your program should compute.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1130>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1392>

[3] <http://acm.uva.es/archive/nuevportal/data/problem.php?p=2201>

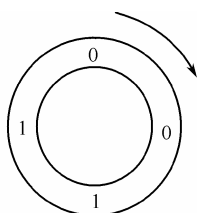


Figure 8-6

k	00110011...	$o(2, k)$
0	00	0
1	01	1
2	11	3
3	10	2

## Input

The input consists of several test cases. For each test case, there will be a line containing two integers  $n$  and  $k$  ( $1 \leq n \leq 15$ ;  $0 \leq k < 2^n$ ). The end of the input file is indicated by a line containing two zeros. Don't process that line.

## Output

For each test case, output  $o(n; k)$  on a line by itself.

## Sample Input

```
2 0
2 1
2 2
2 3
0 0
```

## Sample Output

```
0
1
3
2
```

## Problem Source

Mid-Central European Regional Contest 2000

### 【题目大意】

Ouroboros 是古埃及一条神奇的蛇。它把尾巴咬在嘴里并不断吞食本身。

Ouroboros 数是  $2^n$  位二进制数，并能表示  $0 \sim (2^n - 1)$  之间的数字。产生数字的方法如下：给出 Ouroboros 数，将它的  $2^n$  位绕在一个圆圈上。然后，依次将下一位作为起点，就可以得到  $2^n$  组  $n$  位的数。这样的圆周叫做数  $n$  的 Ouroboros 圆周。对于每个数  $n$ ，只需关注最小的 Ouroboros 数。

例如： $n = 2$ ，有 4 个 Ouroboros 数，分别是 0011; 0110; 1100 和 1001，最小的数是 0011。如图 8-6 所示是 0011 的 Ouroboros 圆周。

表格描述了函数  $o(n; k)$ ，它是计算数  $n$  位最小 Ouroboros 数的 Ouroboros 圆周的第  $K$  个数。编程实现该函数。

## 输入格式

输入有多组测试例。每组测试例一行，是两个整数  $n$  和  $k$  ( $1 \leq n \leq 15$ ;  $0 \leq k < 2^n$ )。当一行是两个 0 时，输入结束，不需要处理。

## 输出格式

对每组测试例输出一行，是  $o(n; k)$ 。

## 【算法分析】

关于 Ouroboros 的资料，在网站上有很多，推荐阅读：

<http://en.wikipedia.org/wiki/Ouroboros>

<http://www.crystalinks.com/ouroboros.html>

如何产生 Ouroboros 数，在网站上讨论的就很少。在博客“暑假训练之记录<sup>[1]</sup>”中，编程产生了  $n \leq 15$  的所有 Ouroboros 数，相当于打表输出。

这是一个单向欧拉路，在 Baidu 中搜索“有向欧拉图”就对了，推荐阅读：

[http://class.htu.cn/lisanshuxue/neirong/7\\_4.htm](http://class.htu.cn/lisanshuxue/neirong/7_4.htm)

属于离散数学的内容：欧拉图与哈密尔顿图。例题《计算机鼓轮的设计》，图中存在一条欧拉回路，采用邻接边标记法（第一条边的后  $n-1$  位二进制数与第二条边的前  $n-1$  位二进制数相同），产生一个二进制数序列，正是  $n=4$  位的 Ouroboros 数，如图 8-7 所示。

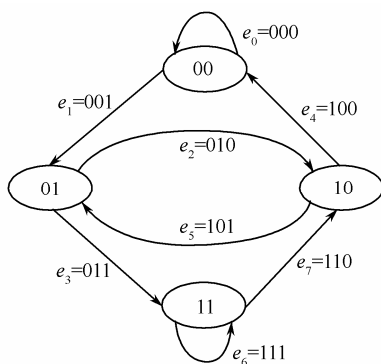


图 8-7  $n=3$  的欧拉图

为节省篇幅，下面以  $n=3$  的欧拉图为例（ $n=4$  的欧拉图，请参考该网站）：

边的生成规则：设结点  $N$  的值是  $a_1a_2$ ，则从结点  $N$  引出的两条边是： $a_1a_20$  和  $a_1a_21$ ，边  $a_1a_20$  指向结点  $a_20$ ，边  $a_1a_21$  指向结点  $a_21$ 。

给定有向图  $G$ ，通过图中每边一次且仅一次的一条单向路（回路），称作单向欧拉路（回路），即一笔画问题。

欧拉回路的构造：从  $e_0$  开始，对结点  $N$ ，选择二进制值小的出度方向前进，并且确保每边走一次且仅走一次。因此，图 8-7 欧拉图的欧拉路是： $e_0e_1e_2e_5e_3e_6e_7e_4$ 。当  $e_2$  走到结点 10 时，不能直奔  $e_4$ ，那其他边就走不成了。当  $e_3$  走到结点  $e_{11}$  时，就直奔边  $e_7e_4$ 。边  $e_7$  是边  $e_6$  去掉最高位然后低位补 0，边  $e_4$  是边  $e_7$  去掉最高位然后低位补 0。边  $e_6$  是结点 11 的自循环结点。再

[1] <http://www.cppblog.com/bnugong/archive/2008/07/15/56217.html?opt=admin>

采用邻接边标记法，就得到了  $n = 3$  位的 Ouroboros 数，如图 8-8 (a) 所示。

在构造欧拉回路时，对每个结点，总是选择二进制值小的出度方向前进（当然要确保每边走一次且仅走一次），即使用了贪心策略，所以这样构造的 Ouroboros 数一定是最小的 Ouroboros 数。

当  $n = 3$  和  $n = 4$  时，Ouroboros 圆周如图 8-8 (b) 所示：

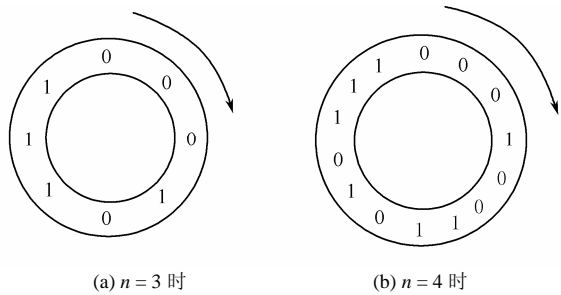


图 8-8 Ouroboros 圆周

估计输入数据不多，本题是每读取一组测试例，计算一次  $o(n; k)$ 。为了计算 Ouroboros 圆周的第  $K$  个数，就得从  $e_0$  开始，逐步构造欧拉路，直到第  $k$  步为止。

(1) 数据结构

$n$  位最小 Ouroboros 数的 Ouroboros 圆周的第  $k$  个数：

```
int n, k;
```

构造欧拉路的辅助数组：

```
char snake[16384];
```

其功能是确定产生下一个数的最低位。该单元是 1，下一个数的最低位是 1，否则是 0。

Ouroboros 圆周的一个数：

```
int number;
```

(2) 建立数组 snake 的初值

令  $mask = 2^n - 1$ ，显然  $mask$  正是结点为全 1 的值。初值  $j = mask$ ，从  $j$  单元起，所有  $j = (j < 1) \& mask$  的单元都是 1，直至  $j = 0$ 。从图 8-7 看出，这些结点，正是欧拉路从全 1 的结点直奔终点的路径，在初始构造欧拉路时，必须在低位先构造 1，才能构造完整的欧拉路。

当  $n = 2$  时，数组 snake 的初值如下：

下标	0	1	2	3
值	0	1	0	0

当  $n = 3$  时，数组 snake 的初值如下：

下标	0	1	2	3	4	5	6	7
值	0	0	1	1	0	0	0	0

数组 snake 的作用是：如果单元  $i$  的值为 1，则根据  $i$  产生的下一个数的低位为 1，否则为 0。由于该单元的值已经用过，则需立即取反。



(3) 构造欧拉路, 计算 Ouroboros 圆周的第  $k$  个数

从  $e_0$  开始, 逐步构造欧拉回路, 直至第  $k$  个数  $number$ 。将当前位置记为:

```
int pos;
```

当前结点记为:

```
int node;
```

构造当前边:  $number = (node \ll 1)$ , 低位是  $snake[node]$  的值。同时  $snake[node]$  取反。

显然, 如果  $pos=k$ , 则  $number$  就是答案。

如果  $node=mask$  时, 就是全 1 的结点, 构造自身回路:  $number = (node \ll 1) + 1$ , 然后直接计算:

```
number = (number & mask) << 1;
```

产生下一个  $number$  (即最低位全部是 0), 直至  $pos=k$ 。

## 【程序代码】

---

程序名称:	zju1130.cpp
题 目:	Ouroboros Snake
提交语言:	C
运行时间:	0ms
运行内存:	160KB

---

```
#include <stdio.h>
#include <memory.h>

int main()
{
    int n, k;                //函数 o(n; k) 的输入参数
    char snake[16384];       //构造欧拉路的辅助数组
    int mask;                //结点为全 1 的结点的值
    int pos;                 //当前位置
    int node;                //当前结点
    int number;              //Ouroboros 圆周的一个数, 当前边
    int i, j;
    while (scanf("%d %d", &n, &k) && (n || k))
    {
        if(n == 1)
        {
            printf("%d\n", k);
            continue;
        }
        mask = (1 << (n-1)) - 1;
        memset(snake, 0, sizeof(snake));
        //建立数组 snake 的初值
        //从结点值为全 1 的结点 mask 奔向终点 (起点) 的沿途结点
        j = mask;
        for (i = 1; i < n; i++)
```

```

{
    snake[j] = 1;
    j = (j<<1) & mask;
}
//从  $e_0$  开始, 逐步构造欧拉回路, 计算 Ouroboros 圆周的第 k 个数
pos = 0;
node = 0;
while(1)
{
    //是全 1 的结点, 构造自身回路
    if (node == mask)
    {
        number = (node<<1)+1;
        break;
    }
    //由当前结点 node, 构造当前边 number
    number = (node<<1);
    //number 的低位就是 snake[node] 的值
    if (snake[node]) number++;
    //该单元需立即取反 (0 或者 1 只能用 1 次)
    snake[node] = !snake[node];
    //去掉 number 的高位, 构造下一个结点
    node = number & mask;
    //已经计算出答案
    if (pos == k) break;
    pos++;
}
//在 node = mask 之后的结点, 低位全部是 0
while (pos != k)
{
    number = (number & mask)<<1;
    pos++;
}
//输出结果
printf("%d\n", number);
}
return 0;
}

```

# ZJU1134-Strategic Game<sup>[1、2、3]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768KB

---

Bob enjoys playing computer games, especially strategic games, but sometimes he cannot find the solution fast enough and then he is very sad. Now he has the following problem. He must defend a medieval city, the roads of which form a tree. He has to put the minimum number of soldiers on the nodes so that they can observe all the edges. Can you help him?

Your program should find the minimum number of soldiers that Bob has to put for a given tree.

The input file contains several data sets in text format. Each data set represents a tree with the following description:

- the number of nodes
- the description of each node in the following format  
node\_identifier:(number\_of\_roads)node\_identifier1 node\_identifier2 ...  
node\_identifier  
or  
node\_identifier:(0)

The node identifiers are integer numbers between 0 and  $n-1$ , for  $n$  nodes ( $0 < n \leq 1500$ ). Every edge appears only once in the input data.

For example for the tree(Figure 8-9):

the solution is one soldier ( at the node 1).

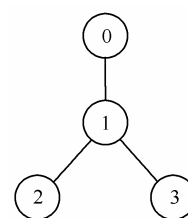


Figure 8-9

The output should be printed on the standard output. For each given input data set, print one integer number in a single line that gives the result (the minimum number of soldiers). An example is given in the following table:

## Input

```
4
0: ( 1 ) 1
1: ( 2 ) 2 3
2: ( 0 )
3: ( 0 )
5
3: ( 3 ) 1 4 2
1: ( 1 ) 0
2: ( 0 )
```

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1134>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1463>

[3] <http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2038>

0: (0)

4: (0)

## Output

1

2

## Problem Source:

Southeastern Europe 2000

### 【题目大意】

Bob 喜欢玩计算机游戏, 尤其是智力游戏。但他有时因为找不到快速的办法而忧郁。现在, 他有一个问题: 他要防卫一个中世纪的城市, 而城市里的道路构成树的形状。除了在树结点处放置最少的士兵外, 他还要确保这些士兵能观察到所有的边。你能帮助他吗?

对于给定的树, 你的程序计算出 Bob 最少应该放置多少个士兵。

输入几组数据。每组数据都构成一棵树, 描述如下:

- 结点的数量

- 每个结点的格式

node\_identifier: (number\_of\_roads) node\_identifier1 node\_identifier2 ... node\_identifier

or

node\_identifier: (0)

结点标识符是 0 至  $n-1$  之间的整数,  $n$  个结点 ( $0 < n \leq 1500$ )。每条边只出现一次。

示例 (图 8-9):

答案是只需要一个士兵 (结点 1 处)。

对每组输入数据输出一个整数, 占一行, 是最少士兵数。

### 【算法分析】

本题的图是一棵树。题目要求计算顶点覆盖数。关于图论的覆盖问题, 参看维基百科:  
<http://zh.wikipedia.org>。

图的覆盖是一些顶点 (或边) 的集合, 使得图中的每一条边 (每一个顶点) 都至少接触集合中的一个顶点 (边)。寻找最小的顶点覆盖问题称为顶点覆盖问题, 它是一个 NP 完全问题。顶点覆盖和边覆盖分别与独立集合和匹配问题有关。在参考书目[1]中, 9.4.2 节, 顶点覆盖问题的近似算法中, 给出了近似最优的顶点覆盖问题算法。

定义: 图  $G$  的顶点覆盖是一个顶点集合  $V$ , 使图  $G$  中的每一条边都接触  $V$  中的至少一个顶点。我们称集合  $V$  覆盖了  $G$  的边。最小顶点覆盖是用最少的边来覆盖给定的顶点。顶点覆盖数  $\tau$  是最小顶点覆盖的大小。

本题可以用贪心算法和动态规划算法实现, 这里采用动态规划算法 (也称为树形动态规划)。关于贪心算法, 可参考网上的相关博客。

(1) 数据结构

采用类似于前向星<sup>[1]</sup>的数据结构存储图。使用数组pos表示树的层次：

```
#define maxV 1501
int pos[maxV];
```

使用数组 size 表示每个结点的子结点个数：

```
int size[maxV];
```

使用结构体数组表示边：

```
struct Edge
{
    int x, y;
}edge[maxV];
```

设所有结点编号的和为 node，所有子结点编号的和为 leaf，由于根结点编号 root 不会出现在子结点编号中，所以根结点编号为：

```
int root = node-leaf;
```

对于样例数据 2，图的结构如图 8-10 所示：

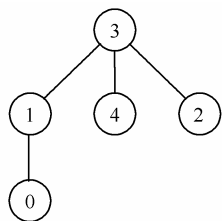


图 8-10 样例数据 2 的图结构

各个数组的值如下所示：

	0	1	2	3	4
数组 pos	5	4	5	1	5
数组 size	0	1	0	3	0

		0	1	2	3
数组 edge	x	3	3	3	1
	y	1	4	2	0

从数组 pos 看出，结点 3 是第 1 层；结点 1 是第 4 层；结点 0、4 和 2 是第 5 层，属于同一层。数值相同的结点在同一层，数值越小，离根结点越近。

从数组 size 看出，结点 1 有 1 个子树，结点 3 有 3 个子树，其余结点没有子树。

各个结点编号的和 node=10，子结点编号的和 leaf=7，则根结点的编号为

```
root = node-leaf = 10-7 = 3
```

(2) 读取数据

本题的原始数据中夹有“:”和“()”，数据的读取比较麻烦。采用 C 语言的格式读取方法，数据的读取就很容易。

[1] <http://baike.baidu.com/view/2078640.htm>

设顶点编号为  $u$ ，子结点个数为  $num$ ，则 C 语言的格式读取语句为：

```
scanf("%d:(%d)", &u, &num );
```

### (3) 树形动态规划算法

设  $f_1[i]$  表示结点  $i$  在最小覆盖集的情况下，以  $i$  为根的子树的最小覆盖集中元素的个数；

设  $f_2[i]$  表示结点  $i$  不设置在最小覆盖集的情况下，以  $i$  为根的子树的最小覆盖集中元素的个数（ $i$  为结点编号  $0 \sim n-1$ ）。

则状态转移方程如下。

当  $i$  不是叶子结点时：

$$f_1[i] = \sum_{k=j_1, j_2, \dots, j_m} \min(f_1[k], f_2[k]) + 1$$

$$f_2[i] = \sum_{k=j_1, j_2, \dots, j_m} f_1[k]$$

式中： $j_1, j_2, \dots, j_m$  都是  $i$  的子结点。

当  $i$  是叶子结点时：

$$f_1[i] = 1$$

$$f_2[i] = 0$$

那么最后的结果就是  $\min(f_1[root], f_2[root])$ 。

### 【程序代码】

程序名称：	zju1134.c
题    目：	Strategic Game
提交语言：	C
运行时间：	160ms
运行内存：	192KB

```
#include <stdio.h>
#include <string.h>

#define maxV 1501

int pos[maxV];           //表示树的层次
int size[maxV];          //每个结点的子结点个数
int f1[maxV], f2[maxV];  //用于动态规划的数组

struct Edge
{
    int x, y;
}edge[maxV];             //存储边的数组

//动态规划算法的实现，形参是当前的结点编号
void dp( int now )
```

```

{
    f1[now] = 1;
    f2[now] = 0;
    if (size[now]==0) return;           //叶子结点
    int v;
    //实现递推公式
    int i = pos[now];
    while(edge[i].x ==now)
    {
        v = edge[i++].y;
        dp( v );
        f1[now] += f1[v]<f2[v]?f1[v]:f2[v];
        f2[now] += f1[v];
    }
}

int main()
{
    int i, j;
    int u, v, num;
    int node, leaf;                   //结点的和, 子结点的和
    int V;                             //结点的数量
    while( scanf("%d", &V )!=EOF )
    {
        int idx = 1;
        node = leaf = 0;
        for( i=1; i<=V; i++ )
        {
            scanf("%d:(%d)", &u, &num );           //采用了格式化参数
            node +=u;
            pos[u] = idx;                           //构造树的层次数组
            size[u] = num;                           //结点 u 的子结点数量
            //构造边
            for( j=1; j<=num; j++ )
            {
                scanf("%d", &v );
                leaf +=v;
                edge[idx].x = u;
                edge[idx++].y = v;
            }
        }
    }
}

```

```

        int root = node-leaf;                //计算根结点编号
        dp(root);                          //调用动态规划算法
        //计算最优值
        printf("%d\n", f1[root]<f2[root]?f1[root]:f2[root]);
    }
    return 0;
}

```

## ZJU1137-Girls and Boys<sup>[1、2、3]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768 KB

---

In the second year of the university somebody started a study on the romantic relations between the students. The relation "romantically involved" is defined between one girl and one boy. For the study reasons it is necessary to find out the maximum set satisfying the condition: there are no two students in the set who have been "romantically involved". The result of the program is the number of students in such a set.

The input contains several data sets in text format. Each data set represents one set of subjects of the study, with the following description:

- the number of students
- the description of each student, in the following format

student\_identifier:(number\_of\_romantic\_relations) student\_identifier1 student\_identifier2  
student\_identifier3 ...

or

student\_identifier:(0)

The student\_identifier is an integer number between 0 and  $n-1$ , for  $n$  subjects.

For each given data set, the program should write to standard output a line containing the result.

An example is given in Figure 1.

### Input

```

7
0: (3) 4 5 6
1: (2) 4 6
2: (0)
3: (0)
4: (2) 0 1
5: (1) 0

```

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1137>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1466>

[3] <http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2041>



```

6: (2) 0 1
3
0: (2) 1 2
1: (1) 0
2: (1) 0

```

## Output

```

5
2

```

## Problem Source:

Southeastern Europe 2000

### 【题目大意】

大学第二年，有人进行了一项同学之间恋爱关系的调查。“恋爱”关系的对象是一个女孩和一个男孩。出于研究的目的，需要调查找出符合下列条件的最大数：在集合中，任何两个学生都没有“恋爱关系”。程序的结果是这个集合中学生的人数。

输入有多组测试数据。每组数据表示一次调查，描述如下：

- 学生的人数
- 每位学生的描述

`student_identifier` 是一个整数，在 0 和  $n-1$  之间，共  $n$  行。

对每组测试数据，程序输出相应结果，占一行。

### 【算法分析】

本题是一道经典的二分图匹配算法的题目。二分图又称作二部图，是图论中的一种特殊模型，在离散数学、组合数学和算法分析与设计等相关书籍中都有介绍。在互联网上有很多这方面的资料，建议读者参考：百度百科<sup>[1]</sup>，相关的博客<sup>[2, 3]</sup>。

设  $G=(V,E)$  是一个无向图，如果顶点  $V$  可分割为两个互不相交的子集  $(A,B)$ ，并且图中的每条边  $(i,j)$  所关联的两个顶点  $i$  和  $j$  分别属于这两个不同的顶点集  $(i \in A, j \in B)$ ，则称图  $G$  为一个二分图。

如图 8-11 所示。

给定一个二分图  $G$ ，在  $G$  的一个子图  $M$  中， $M$  的边集中任意两条边都不依附于同一个顶点，则称  $M$  是一个匹配。选择这样的边数最大的子集称为图的最大匹配问题(Maximal Matching Problem)。

增广路的定义（也称增广轨或交错轨）：

若  $P$  是图  $G$  中一条连通两个未匹配顶点的路径，并且属  $M$  的边和不属  $M$  的边（即已匹配和待匹配的边）在  $P$  上交替出现，则称  $P$  为相对于  $M$  的一条增广路径。

由增广路的定义可以推出下述三个结论：

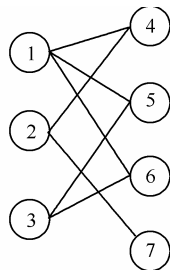


图 8-11 二分图示例

[1] <http://baike.baidu.com/view/501081.htm>

[2] <http://imlazy.ycool.com/post.1603708.html>

[3] <http://acm.zjnu.cn/show.asp?tab=arithmetic&id=82>

①  $P$  的路径长度必定为奇数，第一条边和最后一条边都不属于  $M$ 。

②  $P$  经过取反操作可以得到一个更大的匹配  $M'$ 。

③  $M$  为  $G$  的最大匹配当且仅当不存在相对于  $M$  的增广路径。

求二分图最大匹配，可以用最大流算法或者匈牙利算法。这里使用匈牙利算法。

用增广路求最大匹配，称作匈牙利算法，是匈牙利数学家 Edmonds 于 1965 年提出。

算法思想：

① 置  $M$  为空；

② 找出一条增广路径  $P$ ，通过取反操作获得更大的匹配  $M'$  代替  $M$ ；

③ 重复②操作直到找不出增广路径为止。

本题中只有男生和女生之间存在“恋爱”关系，这个图刚好是一个二分图，题目的答案就是人数  $n$  减去这个二分图的最大匹配数。

问题是题目并没有给出哪些人是男生，哪些是女生，也就是我们不知道那些顶点属于左图，哪些属于右图。如果我们直接对原图进行二分匹配，即左图和右图都包含所有的人，那么匹配的结果应该是正常匹配的 2 倍，所以我们根本就不需要判断男生女生，直接对原图进行匹配。

(1) 数据结构

由于男女之间的关系非常有限，所以采用类似于邻接表的邻接矩阵：

```
int g[M][20];
```

对于样例数据 1，邻接矩阵的值如下：

学生编号	关系数	每个关系			
	第 0 列	第 1 列	第 2 列	第 3 列	
0	3	4	5	6	
1	2	4	6	0	
2	0	0	0	0	
3	0	0	0	0	
4	2	0	1	0	
5	1	0	0	0	
6	2	0	1	0	

(2) 匈牙利算法的实现

是通过函数 `int bipartite()` 实现的。

● 数据结构：

使用数组保存匹配，计算结束时是最大匹配：

```
int match[M];
```

使用数组保存已经搜索过的结点：

```
char used[M];
```

● 匈牙利算法实现框架：

① 初始时最大匹配为空；

② 对二分图左边的每个点  $i$ ，从点  $i$  出发寻找增广路径。如果找到，则把它取反（即增加了匹配数）。

算法实现中使用了一条重要的定理：

如果从一个点  $A$  出发，没有找到增广路径，那么无论再从别的点出发找到多少增广路径

来改变现在的匹配, 从 A 出发都永远找不到增广路径。因此每个结点只搜索一次。

### ● 寻找增广路径

是通过函数 `int dfs(int stu)` 实现的。

从顶点 `stu` 出发, 用深度优先的策略寻找增广路。并在找到之后, 利用递归来修改匹配。

### 【程序代码】

---

程序名称:	zju1137.c
题    目:	Girls and Boys
提交语言:	C
运行时间:	230ms
运行内存:	200KB

---

```
#include<stdio.h>
#include<memory.h>
#define M 500

int n,m; //学生人数, 每个学生的关系数
int match[M]; //最大匹配
char used[M]; //结点是否使用过
int g[M][20]; //二分图的邻接矩阵

//从结点 stu 出发, 用深度优先的策略寻找增广路
int dfs(int stu)
{
    int i;
    int x, t;
    //与结点 stu 匹配的每个结点
    for(i=1; i<=g[stu][0]; i++)
    {
        x = g[stu][i]; //与结点 stu 匹配的 1 个结点
        if(!used[x]) //如果没有用过
        {
            used[x] = 1; //标记为用过
            t = match[x]; //与 x 匹配的结点
            match[x] = stu; //修改匹配
            if(t!=-1||dfs(t)) return 1; //t 是终点或者从 t 出发的一条增广路
            match[x] = t; //为后面的搜索恢复状态
        }
    }
    return 0;
}
```

```

//匈牙利算法的实现
int bipartite()
{
    int stu;
    int res = 0; //匹配数
    memset(match, -1, sizeof(match));
    //对二分图左边的每个点
    for(stu=0; stu<n; stu++)
    {
        memset(used, 0, sizeof(used));
        if(dfs(stu)) res++; //找到了增广路径
    }
    return res;
}

int main()
{
    int i, j;
    int a, b;
    while(scanf("%d",&n)!=EOF)
    {
        //构造邻接矩阵
        memset(g, 0, sizeof(g));
        for(i=0; i<n; i++)
        {
            scanf("%d: (%d)", &a, &m);
            g[a][0] = m; //关系的数量
            for(j=1; j<=m; j++)
            {
                scanf("%d", &b);
                g[a][j] = b; //构造每个关系
            }
        }
        //注意匹配数要除以 2
        printf("%d\n",n-bipartite()/2);
    }
    return 0;
}

```

# ZJU1140-Courses<sup>[1、 2、 3]</sup>

---

Time Limit: 10 Seconds

Memory Limit: 32768KB

---

Consider a group of  $N$  students and  $P$  courses. Each student visits zero, one or more than one courses. Your task is to determine whether it is possible to form a committee of exactly  $P$  students that satisfies simultaneously the conditions:

- every student in the committee represents a different course (a student can represent a course if he/she visits that course)
- each course has a representative in the committee

Your program should read sets of data from a text file. The first line of the input file contains the number of the data sets. Each data set is presented in the following format:

```
P N
Count1 Student1 1 Student1 2 ... Student1 Count1
Count2 Student2 1 Student2 2 ... Student2 Count2
.....
CountP StudentP 1 StudentP 2 ... StudentP CountP
```

The first line in each data set contains two positive integers separated by one blank:  $P$  ( $1 \leq P \leq 100$ ) - the number of courses and  $N$  ( $1 \leq N \leq 300$ ) - the number of students. The next  $P$  lines describe in sequence of the courses . from course 1 to course  $P$ , each line describing a course. The description of course  $i$  is a line that starts with an integer Count  $i$  ( $0 \leq \text{Count } i \leq N$ ) representing the number of students visiting course  $i$ . Next, after a blank, you'll find the Count  $i$  students, visiting the course, each two consecutive separated by one blank. Students are numbered with the positive integers from 1 to  $N$ .

There are no blank lines between consecutive sets of data. Input data are correct.

The result of the program is on the standard output. For each input data set the program prints on a single line "YES" if it is possible to form a committee and "NO" otherwise. There should not be any leading blanks at the start of the line.

An example of program input and Output:

## Input

```
2
3 3
3 1 2 3
2 1 2
1 1
```

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1140>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1469>

[3] <http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2044>

```
3 3
2 1 3
2 1 3
1 1
```

## Output

```
YES
NO
```

## Problem Source

Southeastern Europe 2000

### 【题目大意】

考虑  $N$  个学生和  $P$  门课程。每个学生可以不选课程，或者选修一门或多门课程。编程任务：确定是否刚好由  $P$  个学生构成一个委员会，同时满足下列条件：

- 委员会里的每位学生代表不同的课程（如果他/她选修这门课，则这位学生可以代表这门课）；
- 每门课在委员会里都有一位代表。

输入有多组测试数据。第一行是测试数据组数。每组测试数据格式如下：

第一行两个正整数： $P$  ( $1 \leq P \leq 100$ ) 是课程数和  $N$  ( $1 \leq N \leq 300$ ) 是学生数。接着的  $P$  行是按课程顺序描述各门课程，从课程 1 到课程  $P$ ，每行描述一门课程。第  $i$  门课程的开头是 Count  $i$  ( $0 \leq \text{Count } i \leq N$ )，表示选修这门课程的学生人数。然后，一个空格和选修该课程的 Count  $i$  个学生，每两个学生之间是一个空格。学生编号由  $1 \sim N$  的整数表示。

连续的组与组之间没有空行。输入数据确保是正确的。

如果该组数据能组成一个委员会，那么输出“YES”，否则输出“NO”，占一行。每行开头没有空格。

### 【算法分析】

本题的算法与 ZJU1137-Girls and Boys 是一样的，请读者参考那里的算法说明。

区别在于：二分图的左边是课程，右边是学生，找到的最大匹配是所有的课程各自对应的一个学生，如果学生的人数是  $P$ ，则输出输出“YES”，否则输出“NO”。

### 【程序代码】

---

程序名称:	zju1140.c
题 目:	Courses
提交语言:	C
运行时间:	230ms
运行内存:	304KB

---

```
#include <stdio.h>
#include <memory.h>
```

```
#define MAXN 301
int match[MAXN];
```

```
//最大匹配
```

```

int used[MAXN]; // 结点是否使用过
int g[MAXN][120]; // 二分图的邻接矩阵

// 从结点 course 出发，用深度优先的策略寻找增广路
int dfs(int course){
    int x, t;
    int i;
    for (i = 1; i<=g[course][0]; ++i){
        x = g[course][i];
        if (!used[x]){
            t = match[x];
            match[x] = course;
            used[x] = 1;
            if (t == 0 || dfs(t)) return 1;
            match[x] = t;
        }
    }
    return 0;
}

int main(){
    int cases; // 测试例数
    scanf("%d", &cases);
    while (cases--){
        int i, j;
        // 构造邻接矩阵
        int n, p; // 学生人数，课程数
        scanf("%d%d", &p, &n);
        for (i = 1; i <= p; ++i){
            scanf("%d", &g[i][0]); // 该课程的选修人数
            for (j = 1; j<=g[i][0]; ++j)
                scanf("%d", &g[i][j]);
        }
        // 匈牙利算法的实现
        memset(match, 0, sizeof(match));
        for (i = 1; i <= p; ++i){
            memset(used, 0, sizeof(used));
            dfs(i);
        }
        // 计算最大匹配的路径数量
        int res = 0;
        for (i = 1; i <= n; ++i){
            if (match[i]>0) ++res;
        }
        // 输出结果
        if (res == p) printf("YES\n");
        else printf("NO\n");
    }
}

```

```

    return 0;
}

```

## ZJU1141-Closest Common Ancestors<sup>[1]</sup>

Time Limit: 10 Seconds

Memory Limit: 32768KB

Write a program that takes as input a rooted tree and a list of pairs of vertices. For each pair  $(u, v)$  the program determines the closest common ancestor of  $u$  and  $v$  in the tree. The closest common ancestor of two nodes  $u$  and  $v$  is the node  $w$  that is an ancestor of both  $u$  and  $v$  and has the greatest depth in the tree. A node can be its own ancestor (for example in Figure 1 the ancestors of node 2 are 2 and 5)

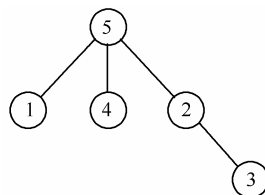


Figure 8-12

The data set starts with the tree description, in the form

```

nr_of_vertices
vertex:(nr_of_successors) successor1 successor2 ... successorn
.....

```

where vertices are represented as integers from 1 to  $n$ . The tree description is followed by a list of pairs of vertices, in the form:

```

nr_of_pairs
(u v) (x y) ...

```

The input contents several data sets (at least one).

Note that white-spaces (tabs, spaces and line breaks) can be used freely in the input.

For each common ancestor the program prints the ancestor and the number of pair for which it is an ancestor. The results are printed on the standard output on separate lines, in to the ascending order of the vertices, in the format: ancestor:times

For example, for the following tree:

the program input and output is:

### Input

```

5
5: (3) 1 4 2
1: (0)
4: (0)
2: (1) 3
3: (0)
6
(1,5) (1,4) (4,2)
(2,3)

```

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1141>



(1,3) (4,3)

## Output

2:1

5:5

## Problem Source

Southeastern Europe 2000

### 【题目大意】

编写程序：输入是一棵有根结点的树和一组顶点对。程序计算每个顶点对  $(u, v)$  最近的祖先。结点  $u$  和  $v$  的最近祖先是树中最深的且既是  $u$  又是  $v$  结点的祖先  $w$ 。一个结点的祖先可能是它自己（例如，图 8-12 中结点 2 的祖先是 2 和 5）。

每组数据的开始是树的描述：

```
nr_of_vertices
vertex: (nr_of_successors) successor1 successor2 ... successorn
.....
```

vertices 是  $1 \sim n$  的整数。树中顶点对描述如下：

```
nr_of_pairs
(u v) (x y) ...
```

输入有多组测试数据（至少一组）。

注意：输入中用到制表符，空格和空行。

程序输出：祖先和对应于该祖先的结点对数。每个结果输出一行，按照结点的升序输出。

### 【算法分析】

本题就是最近公共祖先（Least Common Ancestors）问题。在参考书[5]的专题五中，最近公共祖先问题（LCA）里面有详细的描述。

对于一棵有根树  $T$  的两个结点  $u, v$ ，最近公共祖先  $LCA(T, u, v)$  表示一个结点  $x$ ，满足  $x$  是  $u, v$  的祖先且  $x$  的深度尽可能大。另一种理解方式是把  $T$  理解为一个无向无环图，而  $LCA(T, u, v)$  即  $u$  到  $v$  的最短路上深度最小的点。

对于样例数据，则有：

询问	1	2	3	4	5	6
顶点对	(1,5)	(1,4)	(4,2)	(2,3)	(1,3)	(4,3)
公共祖先	5	5	5	2	5	5

输出的结果是：在全部询问中，祖先结点出现的次数。对于样例数据，结点 2 出现 1 次（输出 2:1），结点 5 出现 5 次（输出 5:5）。

在互联网上，关于最近公共祖先的描述也很多，这里给出一些主要思想。

在参考书[5]中，LCA 问题有离线算法和在线算法两种。

(1) 在线算法

令  $L(u)$  为  $u$  的深度（离根的距离）。设  $L(u) \leq L(v)$ ，则如果  $u$  是  $v$  的父亲， $LCA(u, v) = u$ ，否则  $LCA(u, v) = LCA(u, \text{father}(v))$ 。这样递推的总时间复杂度为  $O(n^2)$ ，即在

$O(n^2)$  的预处理,  $O(1)$  的时间解决了 LCA 问题。

### (2) 离线算法

常用 Tarjan 算法, 是基于深度优先搜索的框架。最开始处理所有的询问, 如果存在询问  $LCA(u, v)$ , 就把  $v$  保存在集合  $Q(u)$  里。初始时, 数组 `checked` 为 `false`。伪代码如下:

```
void LCA(u)
{
    Make-Set(u);
    ancestor[Find-Set(u)] = u
    for (u 的每个孩子 v)
    {
        LCA(v);
        Union(u, v);
        ancestor[Find-Set(u)] = u;
    }
    checked[u] = true
    for (Q(u)中的所有元素 v)
    {
        if (checked[v] = true)
            回答 u 和 v 的最近公共祖先为 ancestor[Find-Set(v)]
    }
}
```

其中 `Make-Set( $u$ )` 表示建立一个集合, 只包含元素  $u$ , 且  $u$  为集合的代表元。`Union( $u, v$ )` 表示把  $u$  和  $v$  所在的集合合并,  $u$  为集合的代表元。`Find-Set( $u$ )` 表示找出集合的代表元。

在任何时候一个集合里面的元素都形成了一棵树, 每处理完一棵子树就把它并到父亲所在的集合中。该算法执行的是深度优先搜索, 对于处理过 (即 `true`) 的结点  $v$ ,  $v$  当前所在集合的代表元就是  $v$  和当前处理结点  $u$  的 LCA。

### (3) 数据结构

使用数组保存树的结构:

```
int g[801][2];
```

对当前结点  $y$ , 下标 0 保存其父结点编号  $x$ , 下标 1 保存该结点的深度。对于样例数据, 数组的值如下:

当前结点	父结点	深度
1	5	1
2	5	1
3	2	2
4	5	1
5	0	0

### (3) LCA 算法的实现

本题采用在线算法, 由函数 `int cca(int x, int y)` 实现:

- 如果结点  $x$  的深度大于结点  $y$  的深度, 则沿着  $x$  的父结点往上递推, 达到与结点  $y$  同样的深度;

● 如果结点  $y$  的深度大于结点  $x$  的深度, 则沿着  $y$  的父结点往上递推, 达到与结点  $x$  同样的深度;

● 如果结点  $x$  与结点  $y$  相等, 则找到了 LCA; 否则结点  $x$  和  $y$  沿着各自的父结点往上升一级, 直到根结点。

将结点成为 LCA 的次数存储在数组中:

```
int ans[801];
```

数组的下标对应结点的编号, 按下标依次输出, 刚好是结点编号的升序。

## 【程序代码】

---

程序名称:	zju1141.c
题 目:	Closest Common Ancestors
提交语言:	C
运行时间:	120ms
运行内存:	164KB

---

```
#include <stdio.h>
#include <memory.h>

int g[801][2];                                //保存树的结构
//LCA 算法的实现, 形参是结点对 (x, y)
int cca(int x, int y)
{
    while(g[x][1] > g[y][1])                //使结点 x 与 y 深度相同
        x = g[x][0];
    while(g[y][1] > g[x][1])                //使结点 y 与 x 深度相同
        y = g[y][0];
    while(g[x][1])                          //非根结点
    {
        if(x == y) break;                  //找到了 LCA
        x = g[x][0], y = g[y][0];          //两个结点同时向根方向递推一步
    }
    return x;                               //返回 LCA
}

int main()
{
    int i, j;
    int n, m;
    int x, y;
    char c;
    int ans[801];                            //存储结点成为 LCA 的次数
    while(scanf ("%d", &n)!=EOF)
    {
        memset(ans, 0, sizeof(ans));
```

```

memset(g, 0, sizeof(g));
//构造树的结构
for (i = 0; i < n; i ++)
{
    scanf("%d:(%d) ", &x, &m);
    for(j = 0; j < m; j ++)
    {
        scanf("%d", &y);
        g[y][0] = x;           //父结点
        g[y][1] = g[x][1] + 1; //该结点的深度
    }
}
scanf ("%d\n", &m);
for (i = 0; i < m; i ++)
{
    scanf("(%d,%d)%c", &x, &y, &c);
    //结点对(x, y)的LCA, 被查询的次数
    ans[cca (x, y)] ++;
}
//按结点的升序输出结果
for(i = 1; i <= n; i ++)
    if(ans[i])
        printf ("%d:%d\n", i, ans[i]);
}
return 0;
}

```

## ZJU1150-S-Trees<sup>[1、2、3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

A Strange Tree (S-tree) over the variable set  $X_n = \{x_1, x_2, \dots, x_n\}$  is a binary tree representing a Boolean function  $f: \{0,1\}^n \rightarrow \{0,1\}$ . Each path of the S-tree begins at the root node and consists of  $n+1$  nodes. Each of the S-tree's nodes has a depth, which is the amount of nodes between itself and the root (so the root has depth 0). The nodes with depth less than  $n$  are called non-terminal nodes. All non-terminal nodes have two children: the right child and the left child. Each non-terminal node is marked with some variable  $x_i$  from the variable set  $X_n$ . All non-terminal nodes with the same depth are marked with the same variable, and non-terminal nodes with different depth are marked with

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1150>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1105>

[3] <http://acm.uva.es/p/v7/712.html>

different variables. So, there is a unique variable  $x_{i1}$  corresponding to the root, a unique variable  $x_{i2}$  corresponding to the nodes with depth 1, and so on. The sequence of the variables  $x_{i1}, x_{i2}, \dots, x_{in}$  is called the variable ordering. The nodes having depth  $n$  are called terminal nodes. They have no children and are marked with either 0 or 1. Note that the variable ordering and the distribution of 0's and 1's on terminal nodes are sufficient to completely describe an S-tree.

As stated earlier, each S-tree represents a Boolean function  $f$ . If you have an S-tree and values for the variables  $x_1, x_2, \dots, x_n$ , then it is quite simple to find out what  $f(x_1, x_2, \dots, x_n)$  is: start with the root. Now repeat the following: if the node you are at is labeled with a variable  $x_i$ , then depending on whether the value of the variable is 1 or 0, you go its right or left child, respectively. Once you reach a terminal node, its label gives the value of the function.

On the picture, two S-trees representing the same Boolean function,  $f(x_1, x_2, x_3) = x_1$  and  $(x_2 \text{ or } x_3)$ , are shown. For the left tree, the variable ordering is  $x_1, x_2, x_3$ , and for the right tree it is  $x_3, x_1, x_2$ .

The values of the variables  $x_1, x_2, \dots, x_n$ , are given as a Variable Values Assignment (VVA)

$$(x_1=b_1, x_2=b_2, \dots, x_n=b_n)$$

with  $b_1, b_2, \dots, b_n$  in  $\{0,1\}$ . For instance,  $(x_1 = 1, x_2 = 1, x_3 = 0)$  would be a valid VVA for  $n = 3$ , resulting for the sample function above in the value  $f(1,1,0) = 1$  and  $(1 \text{ or } 0) = 1$ . The corresponding paths are shown bold in the picture.

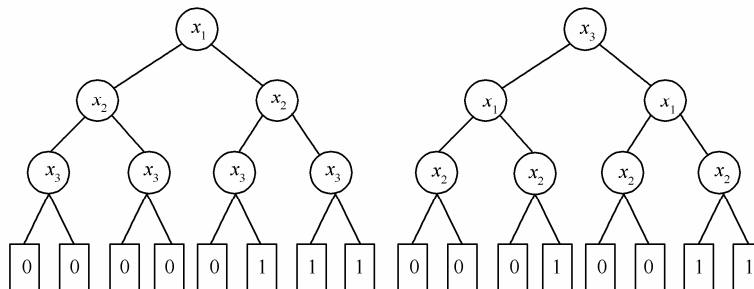


Figure 8-13: S-trees for the  $x_1$  and  $(x_2 \text{ or } x_3)$  function

Your task is to write a program which takes an S-tree and some VVAs and computes  $f(x_1, x_2, \dots, x_n)$  as described above.

## Input

The input contains the description of several S-trees with associated VVAs which you have to process. Each description begins with a line containing a single integer  $n$ ,  $1 \leq n \leq 7$ , the depth of the S-tree. This is followed by a line describing the variable ordering of the S-tree. The format of that line is  $x_{i1} x_{i2} \dots x_{in}$ . (There will be exactly  $n$  different space-separated strings). So, for  $n = 3$  and the variable ordering  $x_3, x_1, x_2$ , this line would look as follows:

$x_3 x_1 x_2$

In the next line the distribution of 0's and 1's over the terminal nodes is given. There will be exactly  $2^n$  characters (each of which can be 0 or 1), followed by the new-line character. The characters are given in the order in which they appear in the S-tree, the first character corresponds to

the leftmost terminal node of the S-tree, the last one to its rightmost terminal node.

The next line contains a single integer  $m$ , the number of VVAs, followed by  $m$  lines describing them. Each of the  $m$  lines contains exactly  $n$  characters (each of which can be 0 or 1), followed by a new-line character. Regardless of the variable ordering of the S-tree, the first character always describes the value of  $x_1$ , the second character describes the value of  $x_2$ , and so on. So, the line

110

corresponds to the VVA ( $x_1=1, x_2=1, x_3=0$ ).

The input is terminated by a test case starting with  $n = 0$ . This test case should not be processed.

## Output

For each S-tree, output the line "S-Tree #j:", where  $j$  is the number of the S-tree. Then print a line that contains the value of  $f(x_1, x_2, \dots, x_n)$  for each of the given  $m$  VVAs, where  $f$  is the function defined by the S-tree.

Output a blank line after each test case.

## Sample Input

3	3	0
x1 x2 x3	x3 x1 x2	
00000111	00010011	
4	4	
000	000	
010	010	
111	111	
110	110	

## Sample Output

```
S-Tree #1:
0011
S-Tree #2:
0011
```

## Problem Source:

Mid-Central European Regional Contest 1999

### 【题目大意】

一个奇怪的树 (S-tree) 对应集合  $X_n = \{x_1, x_2, \dots, x_n\}$ , 是一个二叉树, 表示布尔函数  $f: \{0,1\} \rightarrow \{0,1\}$ 。S-tree 的每条路径都从根结点开始, 路径上有  $n+1$  个结点。每一个 S-tree 的结点都有一个深度, 这是结点本身与根结点之间的结点数 (所以根结点深度为 0)。深度小于  $n$  的结点称为非终结点, 所有非终结点有两个子结点: 右子结点, 左子结点。每个非终结点用集合  $X_n$  中的变量  $x_i$  标识。所有相同深度的非终结点使用相同的变量, 不同深度的非终结点标有不同的变量。所以, 根结点只有唯一的一个变量  $x_{i1}$ , 深度为 1 的结点有唯一的变量  $x_{i2}$ , 依此类推。变量的序列  $x_{i1}, x_{i2}, \dots, x_{in}$  称为变量序列。深度为  $n$  的结点称为终结点, 它们没有

子结点，标有 0 或 1。请注意，变量序列和终结点上 0 和 1 的分布，足以完全描述一个 S-tree。

如前所述，每个 S-tree 代表一个布尔函数  $f$ 。如果你有一个 S-tree 和变量  $x_1, x_2, \dots, x_n$  的值，那么就很容易找出函数  $f(x_1, x_2, \dots, x_n)$ ：从根结点开始，重复下列步骤：如果当前结点标有变量  $x_i$ ，根据变量的值是 1 还是 0，分别去找它的右结点或左结点。一旦你到达一个终结点，则该终结点的标签就是函数的值。

图 8-12:  $x_1 \wedge (x_2 \vee x_3)$  函数的 S-tree

在图片上，两个 S-tree 表示相同的布尔函数， $f(x_1, x_2, x_3) = x_1 \wedge (x_2 \vee x_3)$ 。对左边的树，变量序列为  $x_1, x_2, x_3$ ，而右边的树为  $x_3, x_1, x_2$ 。

变量  $x_1, x_2, \dots, x_n$  的值以变量赋值的方式 (VVA) 给出

$$(x_1=b_1, x_2=b_2, \dots, x_n=b_n)$$

其中  $b_1, b_2, \dots, b_n \in \{0,1\}$ 。例如， $(x_1=1, x_2=1, x_3=0)$  是  $n=3$  时一个有效的 VVA，其函数值  $f(1, 1, 0) = 1 \wedge (1 \vee 0) = 1$ 。相应的路径在上图中用粗线表示。

编程任务：给定一个 S-tree 和一些 VVAs，计算函数  $f(x_1, x_2, \dots, x_n)$ 。

### 输入格式

输入有多组测试数据，包含 S-tree 和相应的 VVAs。每组测试数据，第一行是一个整数  $n$  ( $1 \leq n \leq 7$ )，是 S-tree 的深度。接下来一行就是 S-tree 的变量序列，其格式为  $x_{i1}, x_{i2}, \dots, x_{in}$  (确保有  $n$  个由空格分隔的字符串)。因此， $n=3$  且变量序列为  $x_3, x_1, x_2$  时，这行看起来像这样：

$x_3, x_1, x_2$

接下来一行是终结点上 0 和 1 的分布。确保有  $2^n$  字符 (每个是 0 或 1)，后面是换行符。给出的字符顺序就是它们在 S-tree 中的顺序，第一个字符是 S-tree 中最左边的终结点，最后一个字符是最右边的终结点。

下面一行是一个正整数  $m$ ，是 VVA 的数量。接下来有  $m$  行，每行确保有  $n$  个字符 (每个是 0 或 1)，后面是换行符。不管 S-tree 中的变量序列是怎样的，第一个字符是描述  $x_1$  的值，第二个字符描述  $x_2$  的值，以此类推。因此，下面这行

110

对应于 VVA ( $x_1=1, x_2=1, x_3=0$ )。

当  $n=0$  时输入结束，该测试例不需要处理。

### 输出格式

对于每一个 S-tree，输出行“S-Tree #j:”，其中  $j$  是 S-tree 的序号。然后输出一行，是  $f(x_1, x_2, \dots, x_n)$  的值，对应所给的 VVAs，其中  $f$  是由 S-tree 定义的函数。

每组测试数据后输出一个空行。

### 【算法分析】

题目中已经详细描述了 S-Tree，关键的地方是如何根据 VVA 的值确定终结点的值。

#### (1) 数据结构

存储变量序列的数组：

```
int x[10];
```

存储终结点上 0 和 1 的分布：

```
char terminal[130];
```

S-tree 的深度:

```
int n;
```

需要计算的 VVA 数量:

```
int m;
```

(2) 根据 VVA 的值确定终结点的值

使用变量  $j$  ( $0 \leq j < n$ ) 表示 S-Tree 的深度, 根结点深度为 0。变量  $k$  为终结点序列的序号,  $k=0$  表示最左边的一个终结点。

- 确定  $x_1, x_2, \dots, x_n$  在 VVA 中的序号

根据变量序列的数组,  $x_j$  序号为:  $x[j]-1, 0 \leq j < n$

若变量序列为:  $x_3 x_1 x_2$ , 则它们在 VVA 中的序号分别为: 2 0 1。

- 确定 VVA 对应的终结点

在当前位置  $k$ , 若  $x_j$  (序号为  $x[j]-1$ ) 的值为 0 (沿左结点前进), 位置  $k$  不变; 若为 1 (沿右结点前进), 位置  $k$  的增量  $\Delta k$  为

$$\Delta k = 2^{n-j-1}$$

### 【程序代码】

---

程序名称:	zju1150.c
题 目:	Trees
提交语言:	C
运行时间:	0ms
运行内存:	160KB

---

```
#include<stdio.h>

int main()
{
    int i,j,k;
    int x[10];                //变量序列
    char terminal[130];       //终结点上 0 和 1 的分布
    int n;                    //S-tree 的深度
    int m;                    //计算的 VVA 数量
    char s[10];
    int iCase = 1;            //测试例编号
    while(scanf("%d",&n)!=EOF && n)
    {
        //读取变量序列
        for(i=0; i<n; i++)
        {
            scanf("%s",s);
            x[i] = s[1] - '0';    //取出其中的数字 (1≤n≤7)
        }
        scanf("%s", terminal);
        scanf("%d", &m);
        printf("S-Tree #d:\n",iCase++);
    }
}
```



```

//对每个要计算的 VVA
for(i=1; i<=m; i++)
{
    scanf("%s", s);
    //计算函数 f 的值, k 是终结点的编号
    k=0;
    for(j=0; j<n; j++)
        if(s[x[j]-1]=='1')
            k += 1<<(n-j-1);
    printf("%c", terminal[k]);
}
printf("\n\n");
}
return 0;
}

```

## 第九章 几何和数学题

在本章的题目主要是几何题和数学类题目。由于题目量比较少，将它们归结在一起。

### ZJU1081-Points Within<sup>[1, 2]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768K

---

#### Statement of the Problem

Several drawing applications allow us to draw polygons and almost all of them allow us to fill them with some color. The task of filling a polygon reduces to knowing which points are inside it, so programmers have to colour only those points.

You're expected to write a program which tells us if a given point lies inside a given polygon described by the coordinates of its vertices. You can assume that if a point is in the border of the polygon, then it is in fact inside the polygon.

#### Input Format

The input file may contain several instances of the problem. Each instance consists of: (i) one line containing integers  $N$ ,  $0 < N < 100$  and  $M$ , respectively the number of vertices of the polygon and the number of points to be tested. (ii)  $N$  lines, each containing a pair of integers describing the coordinates of the polygon's vertices; (iii)  $M$  lines, each containing a pair of integer coordinates of the points which will be tested for "withinness" in the polygon.

You may assume that: the vertices are all distinct; consecutive vertices in the input are adjacent in the polygon; the last vertex is adjacent to the first one; and the resulting polygon is simple, that is, every vertex is incident with exactly two edges and two edges only intersect at their common endpoint. The last instance is followed by a line with a 0 (zero).

#### Output Format

For the  $i$ th instance in the input, you have to write one line in the output with the phrase "Problem i:", followed by several lines, one for each point tested, in the order they appear in the input. Each of these lines should read "Within" or "Outside", depending on the outcome of the test. The output of two consecutive instances should be separated by a blank line.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1081>

[2] <http://acm.uva.es/archive/nuevoportal/data/problem.php?p=2392>

## Sample Input

3 1	3 2
0 0	4 4
0 5	3 1
5 0	1 2
10 2	1 3
	2 2
	0

## Sample Output

Problem 1:  
Outside

Problem 2:  
Outside  
Within

## Problem Source

South America 2001

### 【题目大意】

#### 问题描述

一些画图应用程序允许我们绘制多边形，而且几乎所有这些程序允许我们将它们填充颜色。填充一个多边形的任务就简化为：知道在它里面有哪些点，因此程序设计者只需把那些点涂上颜色即可。

编程任务：判断给出的点是否在多边形里，多边形由其顶点的坐标给出。可以假定：如果一个点位于多边形的边界上，也算作位于多边形的里面。

#### 输入格式

输入包含多个实例。每个实例有：

- (1) 第一行包含整数  $N$  ( $0 < N < 100$ ) 和  $M$ ，分别是多边形的顶点数和要测试点的数量；
- (2)  $N$  行，每行是一对整数，描述多边形顶点坐标；
- (3)  $M$  行，每行是一对整数，需要测试是否在该多边形里面。

假设：顶点都是不重复的；连续输入的坐标在多边形中是相邻的顶点；最后一个顶点与第一个顶点相邻；产生的多边形是简单多边形，也就是说，每个顶点与两条边相连，两条边共用一个顶点。一行只有一个 0 时，表示输入结束。

#### 输出格式

对于输入的第  $i$  个实例，输出一行 “Problem i:”。接着是多行，每行按输入顺序输出每个测试点的结果，即 “Within” 或 “Outside”。两个相邻的实例之间有一个空行。

### 【算法分析】

这是一个经典的计算几何问题，请阅读参考书目[5]，《算法艺术与信息学竞赛》，第3章，第3.2.3节，判点在形内形外形上；多面体的情形。主要有两种基本方法：射线法与环顾法。

并对两种方法的基本原理、特殊情况及其解决方法作了详细的介绍。

本题采用射线法。

### (1) 样例分析

如图 9-1 所示。

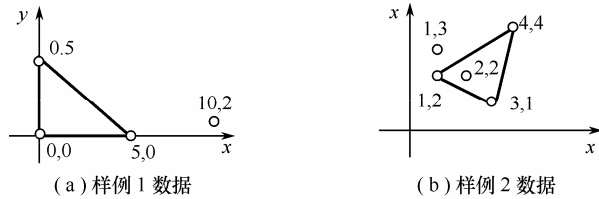


图 9-1 样例数据的结果

### (2) 判断点是否在多边形的一条边上

由函数 inEdge()实现:

```
int inEdge(int x1, int y1)
```

首先判断点  $(x_1, y_1)$  与线段的端点  $(x_j, y_j)$ 、 $(x_{j+1}, y_{j+1})$  之间的夹角, 采用叉乘的方法, 如图 9-12 所示:

$$\begin{aligned}\overline{ab} \times \overline{ac} &= (x_j - x_1) \times (y_{j+1} - y_1) - (x_{j+1} - x_1) \times (y_j - y_1) \\ &= |ab| \cdot |ac| \sin \theta\end{aligned}$$

显然, 结果为 0 时, 夹角为 0, 表示三个点  $(a, b, c)$  共线。

接着判断点  $a$  是否在线段  $bc$  的中间, 还是在线段  $bc$  的外面。这只要判断坐标  $(x_1, y_1)$  是否在坐标  $(x_j, y_j)$  和  $(x_{j+1}, y_{j+1})$  之间, 由函数 order()实现。

### (3) 判断点是否在多边形里面

由函数 inPolygon()实现:

```
int inPolygon(int x1, int y1)
```

采用射线法, 即过点  $P(x_1, y_1)$  沿  $x$  轴正向作一条射线, 根据射线与多边形相交的次数的奇偶性来判断内外: 奇数次在多边形内, 偶数次在多边形外, 如图 9-3 所示。

判断相交的方法: 对每条线段, 只要一个端点 (高点) 高于射线, 另一个端点 (低点) 低于射线 (或在射线上), 且交点在  $P$  点的右侧 ( $x$  轴正向) 即可。

为了判断交点在  $P$  点的右侧, 采用叉积法: 对于一个相交线段, 区别高点  $A$  和低点  $B$ , 线段  $PB$  到线段  $PA$  为右手螺旋, 则交点在  $P$  的右侧, 否则在左侧, 如图 9-4 所示:

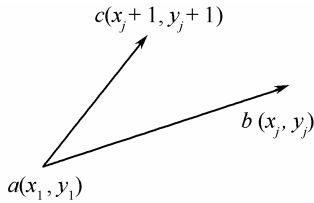


图 9-12 矢量叉乘的计算

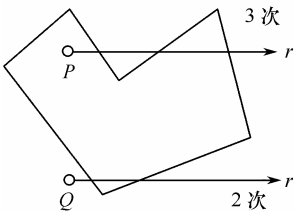


图 9-3 射线法的基本情况

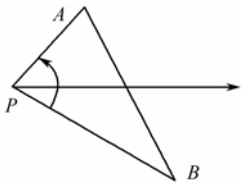


图 9-4 叉积用于判断交点左右

## 【程序代码】

---

程序名称:	zju1081.c
题    目:	Points Within
提交语言:	C++
运行时间:	00:00.00
运行内存:	392KB

---

```
#include <stdio.h>
#include <stdlib.h>

//判断数值  $b$  是否在  $(a, c)$  或  $(c, a)$  的中间
bool order(int a, int b, int c)
{
    return(((a<=b) && (b<=c)) || ((a>=b) && (b>=c)));
}

int x[102], y[102];           //多边形的顶点坐标
int n, m;                     //多边形的顶点数和要测试点的数量

//判断点  $(x_1, y_1)$  是否在线段上
int inEdge(int x1, int y1) {
    int j;
    //对多边形的所有线段, 逐边查找
    for(j=0; j<n; j++)
        //通过叉乘计算夹角, 为 0 时表示三点共线
        if ((x[j]-x1)*(y[j+1]-y1)-(x[j+1]-x1)*(y[j]-y1)==0)
            //判断点是否在线段的中间
            if (order(x[j], x1, x[j+1]))
                if (order(y[j], y1, y[j+1]))
                    return 1;
    return 0;
}

//判断点  $(x_1, y_1)$  是否在多边形里面
int inPolygon(int x1, int y1) {
    //射线与多边形相交的次数
    int count = 0;
    int i, j;
    //两个点不要在水平线上
    for(i=0; i<n; i++)
        if (y[i]!=y1) break;
    int dy1 = y[i]-y1;
    int dx1 = x[i]-x1;
    int dx2, dy2;
    int cross = 0;              //叉积
    //与所有的线段计算相交的次数
```

```

for(j=0; j<n; j++)
{
    //下一个点, 且循环计算
    if ((++i)==n) i=0;
    dx2 = x[i]-x1;
    dy2 = y[i]-y1;
    //与下一个点也不在水平线上
    if (dy2)
    { //计算叉积
        if (cross==0) cross = dx1*abs(dy2) + dx2*abs(dy1);
        //射线与线段相交
        if ((dy1<0 && dy2>0) || (dy1>0 && dy2<0))
            if (cross>0) count++; //交点在右侧时计数
        //准备下一个线段的计算
        dy1 = dy2;
        dx1 = dx2;
        cross = 0;
    }
    else cross = x[i]-x1; //在同一水平线上
}
return count%2; //只需要奇偶数
}

int main()
{
    int x1, y1; //测试点的坐标
    int cases = 1; //测试例数
    while (scanf("%d", &n) && n)
    {
        scanf("%d", &m);
        //读取多边形
        int i;
        for(i=0; i<n; i++)
            scanf("%d%d", &x[i], &y[i]);
        x[n] = x[0];
        y[n] = y[0];
        if (cases!=1) printf("\n");
        printf("Problem %d:\n", cases++);
        //m个测试点
        for(i=0; i<m; i++)
        {
            scanf("%d%d", &x1, &y1);
            //点 ( $x_1$ ,  $y_1$ ) 就在多边形的边界上
            if(inEdge(x1, y1))
            {
                printf("Within\n");
            }
        }
    }
}

```

```

        continue;
    }
    //点 ( $x_1$ ,  $y_1$ ) 是否在多边形里面
    if (inPolygon(x1, y1)) printf("Within\n");
    else printf("Outside\n");
}
}
return 0;
}

```

## ZJU1096-Subway<sup>[1, 2, 3]</sup>

---

Time limit: 1 Seconds    Memory limit: 32768KB

---

Subway trains are meant to move people as quickly, safely, and comfortably as possible from station to station. Although the train drivers' unions may not agree, computer operated trains accomplish these goals more effectively than human operated trains. You are to determine the optimal control strategy to move the train from one station to another within the constraints imposed by safety and comfort considerations, as well as the physical limitations of the train itself.

The parameters to the problem are all positive integers not greater than 1000.

- $d$  - the distance between stations, in metres
- $m$  - the maximum allowable speed of the train, in metres/sec
- $a$  - the maximum absolute acceleration of the train, in metres/sec<sup>2</sup>
- $j$  - the maximum absolute jerk, in metres/sec<sup>3</sup>

The train must be completely stopped at each station and must move in one direction at speeds not exceeding  $m$ . Acceleration can be positive (forward) or negative (backwards) but its absolute value must not exceed  $a$ . The last parameter, jerk, is the rate of change of acceleration in either direction. That is, acceleration cannot increase or decrease at greater than this rate. This parameter prevents toppling the standing passengers.

### Input

There are several test cases. For each test case, standard input has a single line with  $d, m, a, j$ .

### Output

For each test case, standard output should contain a single line giving the minimum time in seconds, to the nearest 0.1 second, required to move between the stations.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1096>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2548>

## Sample Input

1000 70 20 1

## Output for Sample Input

31.7

## Problem Source

University of Waterloo Local Contest 2001.06.02

### 【题目大意】

地铁是一种快捷、安全和舒适的交通工具。虽然火车驾驶员协会可能并不同意这个观点，但是，用计算机操作火车将会比人更有效地实现这些目标。基于安全、舒适和火车本身的考虑，你得确定火车在任何两个站点之间行驶的最佳方案。

问题的参数都是正整数，且不大于 1000。

$d$  为站点之间的距离，单位 m；

$m$  为火车的最大允许速度，单位 m/s；

$a$  为火车的最大绝对加速度，单位  $\text{m/s}^2$ ；

$j$  为最大绝对加速度变化率，单位  $\text{m/s}^3$ 。

每到一个站点，火车都得完全停止并且只能以小于  $m$  的速度朝同一方向行驶。加速度可能是正数（向前）或者负数（向后），但它的绝对数值不能超过  $a$ 。最后一个参数  $\text{jerk}$ ，是加速度在任一方向上的变化率。也就是，加速度不能超过它本身的范围增加或减少，这个参数避免站着的乘客跌倒。

### 输入格式

有几组测试例。每组测试例一行，是  $d$ 、 $m$ 、 $a$ 、 $j$ 。

### 输出格式

对每组测试例，输出一行，以秒为单位输出火车通过两站点的最短时间，精确到十分位。

### 【算法分析】

本题是已知地铁两个站点之间的距离 $d$ 、最大允许速度 $v$ （题目中是 $m$ ）、最大绝对加速度 $a$ 和最大绝对加速度变化率 $j$ ，求火车行驶的最短时间 $t$ 。由于一般很少使用加速度的变化率，所以本题非常具有挑战性。在University of Waterloo Local Contest的竞赛页面<sup>[1]</sup>上，有输入输出数据和标程。该标程使用的公式不易理解，在竞赛时很难想到。下面使用分段计算的方法。

设达到最大加速度  $a$  的时间是  $t_0$ ，则

$$t_0 = a/j$$

设达到最大速度  $v$  所需要的时间是  $t_1$ ，则

$$t_1 = \sqrt{v/j}$$

---

[1] <http://plg1.cs.uwaterloo.ca/~acm00/010602/data/>



由于是左右对称的，下面的加速度和速度图形都只画一半。

(1) 由于距离太短，火车运行时既不能达到最大加速度  $a$ ，也不能达到最大速度  $v$  火车运行时的加速度，如图 9-5 所示。

火车运行时的速度，如图 9-6 所示。

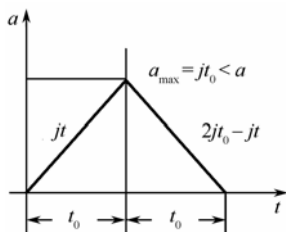


图 9-5 火车运行时的加速度

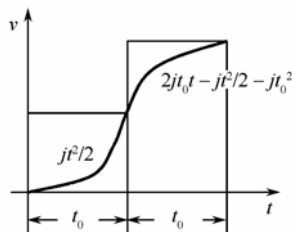


图 9-6 火车运行时的速度图

对第一个  $t_0$  阶段，行驶的路程  $s_1$  是

$$s_1 = \int_0^{t_0} \frac{1}{2} jt^2 dt = \left[ \frac{1}{6} jt^3 \right]_0^{t_0} = \frac{1}{6} jt_0^3$$

对第二个  $t_0$  阶段，行驶的路程  $s_2$  是

$$\begin{aligned} s_2 &= \int_{t_0}^{2t_0} \left( 2jt_0t - \frac{1}{2} jt^2 - jt_0^2 \right) dt \\ &= \left[ jt_0t^2 - \frac{1}{6} jt^3 - jt_0^2t \right]_{t_0}^{2t_0} = \frac{5}{6} jt_0^3 \end{aligned}$$

$$d = 2 (s_1 + s_2)$$

解得

$$d = 2 jt_0^3$$

设火车行驶的最短时间是  $t$ ，则有

$$t = 4t_0 = 4 \times \sqrt[3]{d/(2j)}$$

如果  $t_1 > t_0$ ，说明先达到最大加速度  $a$ ，然后再向最大速度  $v$  加速前进。火车运行时的加速度图，如图 9-7 所示。

设达到最大加速度  $a$  时的速度是  $v_0$ ，则

$$v_0 = j \times t_0^2 / 2$$

火车运行时的速度图，如图 9-8 所示。

设匀加速过程需要的时间是  $t_2$ ，因为加加速度和减加速度的时间段是对称的，则

$$t_2 = (v - 2 \times v_0) / a$$

假设达到最大速度  $v$  之前运行的距离是  $\text{dist}/2$ ，则（推导过程从略）

$$\text{dist} = 2 \times j \times t_0^3 + t_2 \times v + 2 (v - 2 \times v_0) \times t_0$$

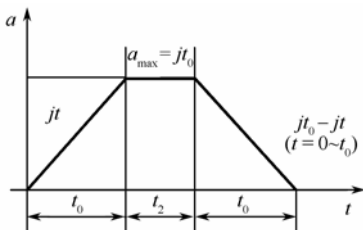


图 9-7 火车运行时的加速度图

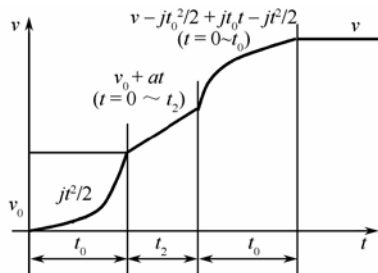


图 9-8 火车运行时的速度图

当  $\text{dist} \leq d$  (就有以最大速度  $v$  匀速运动的过程), 火车行驶的最短时间是:

$$\text{time} = 4 \times t_0 + 2 \times t_2 + (d - \text{dist}) / v$$

如果  $\text{dist} > d$ , 说明没有达到最大速度  $v$ 。火车运行时的速度如图 9-9 所示。

记  $v_1$  为火车以匀加速度运行结束时的速度, 图 9-9

中三段距离的和为  $d/2$ 。

$$v_1 = \frac{a}{2} \left( \sqrt{t_0^2 + \frac{4 \times d}{a}} - t_0 \right)$$

则 
$$\text{time} = 4 \times t_0 + 2 \times \frac{v_1 - v_0}{a}$$

如果  $t_1 < t_0$ , 说明火车没有达到最大加速度  $a$  时, 已经达到了最大速度  $v$ 。如果  $j$  很大, 火车在没有达到最大加速度时就已经达到了最大速度  $v$ , 如图 9-10 所示。

火车运行时的速度, 如图 9-11 所示。

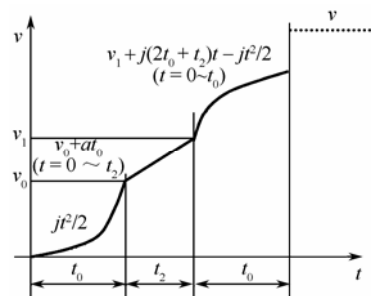


图 9-9 火车运行时的速度图

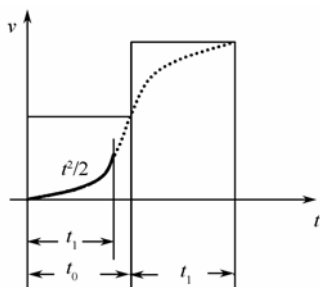


图 9-10 火车运行时的速度图 ( $t_1 < t_0$  的情况)

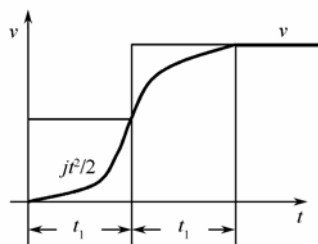


图 9-11 火车运行时的速度图

令:  $\text{dist} = j \times t_1^3$  (匀加速度阶段所走过的路程, 见情况 1), 火车以加速度  $a$  运行时通过的距离。

如果  $\text{dist} \leq d/2$ , 则:

$$\text{time} = 4 \times t_1 + 2 \times \frac{d - 2 \times \text{dist}}{v}$$

## 【程序代码】

---

程序名称:	zju1096.c
题    目:	Subway
提交语言:	C
运行时间:	00:00.00
运行内存:	396KB

---

```
#include <stdio.h>
#include <math.h>

int main()
{
    double d, v, a, j;
    double t0, t1, t2;
    double v0, v1;
    double time;
    double dist;

    while(scanf("%lf%lf%lf%lf", &d, &v, &a, &j)!=EOF)
    {
        t0 = a/j;
        t1 = sqrt(v/j);
        //距离太短, 火车运行时既不能达到最大加速度 a, 也不能达到最大速度 v
        if (t1 > t0 ? j*t0*t0*t0 >= d/2 : j*t1*t1*t1 >= d/2)
            time = pow(d/j/2, 1/3.0)*4;
        else if (t1 > t0) //达到最大加速度 a
        {
            //达到最大加速度 a 时的速度
            v0 = j*t0*t0/2;
            //匀加速过程需要的时间
            t2 = (v-2*v0)/a;
            //达到最大速度 v 之前运行的距离是 dist/2
            dist = 2*j*t0*t0*t0 + t2*v + 2*(v-2*v0)*t0;
            //有以最大速度 v 匀速运动的过程
            if (dist <= d)
                time = t0*4 + t2*2 + (d-dist)/v;
            else
            {
                //没有达到最大速度 v
                //火车以匀加速度运行结束时的速度
                v1 = a*(sqrt(t0*t0+d*4/a)-t0)/2;
                time = t0*4 + 2*(v1-2*v0)/a;
            }
        }
        else
        {
            //没有达到最大加速度 a
            //匀加速度阶段所走过的路程
```

```

        dist = j*t1*t1*t1;
        time = t1*4 + (d-2*dist)/v;
    }
    printf("%0.11f\n", time);
}
return 0;
}

```

## ZJU1104-Leaps Tall Buildings<sup>[1]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

**Special Judge**

---

It's a bird! It's a plane! It's coming right at us!

Although it sometimes seems like it, Superman can't fly (without a plane). Instead, he makes super-human leaps, especially over tall buildings. Since he never knows when he will need to catch a criminal, he can't register flight paths. To avoid hitting planes, he tries to keep his jumps as low to the ground as he can. Given a city-scape as input, find the angle and velocity of Superman's jump that minimizes his maximum altitude.

Recall that gravity provides an acceleration of  $9.8 \text{ m/s}^2$  downwards and the formula for Superman's vertical distance from his starting location is  $d(t) = vt + 0.5 a t^2$  where  $v$  is his initial velocity,  $a$  is his acceleration and  $t$  is time in seconds since the start of the leap.

### Input

Input consists of a sequence of city-scapes, each of the form

```

n
0 d1
h2 d2
.....
h(n-1) d(n-1)
0 dn

```

Superman starts at ground level and leaps  $d_1 + \dots + d_n$  meters, landing at ground level and clearing all of the buildings at heights  $h_2$  to  $h_{(n-1)}$ , each with the given widths.  $n$  will be at most 100.

### Output

Output is the angle and initial velocity that minimizes the height that Superman attains both appearing on the same line. The values should be given to two decimal places and be accurate within 0.01 degrees or m/s, as appropriate.

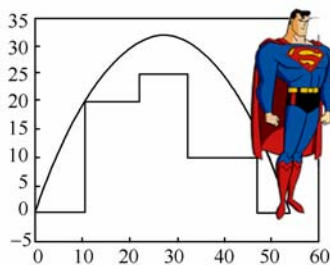
---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1104>

# Sample Input

```
3
0 5
10 5
0 5
5
0 10.5
20 11.5
25 10
10 15
0 7
```

Diagram for Second City-scape



(Not to scale.)

# Sample Output

```
71.57 15.65
67.07 27.16
```

# Problem Source

University of Waterloo Local Contest 1998.06.06

## 【题目大意】

是鸟！是飞机！它正朝我们飞来！

虽然事情有时显得如此奇特，但是超人不能飞（如果没有飞机）。其实，他能腾空飞跃，尤其是跨过高大的建筑物。因为他不知道何时会逮到罪犯，所以他并不能确定飞行路线。为了避免撞到飞机，他尽可能靠近地面飞跃。给出城市景观图，为了能使超人飞跃的最大高度最小，请计算超人飞跃时的角度和速度。

重力加速度为  $9.8\text{m/s}^2$ 。离开初始位置后的高度公式： $d(t) = vt + 0.5at^2$ ， $v$  是超人的初始速度， $a$  是加速度， $t$  是飞跃时间（秒）。

## 输入格式

输入是一些城市景观图，形式如下：

```
n
0 d1
h2 d2
.....
```

$$\begin{matrix} h_{(n-1)} & d_{(n-1)} \\ 0 & d_n \end{matrix}$$

超人从地面飞跃，经过  $d_1 + \dots + d_n$ （米）的不同飞跃段后，最后回到地面，并跨越了所有建筑物，其高度分别是  $h_2 \sim h_{(n-1)}$ ，另一个量就是相应建筑物的宽度。 $n$  最多 100。

### 输出格式

对每一个城市景观图输出一行，是超人起飞时的角度和初始速度，达到飞跃高度尽可能的低。数值保留两位小数，并精确到 0.01（度或 m/s）。

### 【算法分析】

本题是求解物体的斜抛运动，因此需要建立抛物线方程。

如图 9-12 所示，是城市景观图的抛物线图像：

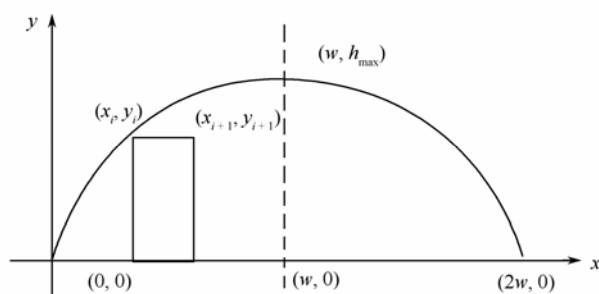


图 9-12 城市景观图的抛物线图像

图中，超人跨越的总宽度为  $2w$ ，最大高度是  $h_{\max}$ ，则最高点的坐标是  $(w, h_{\max})$ 。对建筑物  $i$ ，有两个顶点坐标： $(x_i, y_i)$  和  $(x_{i+1}, y_{i+1})$ ，坐标之间的关系为

$$\begin{aligned} x_{i+1} &= x_i + d_i \\ y_{i+1} &= y_i \end{aligned}$$

其中  $d_i$  是建筑物  $i$  的宽度。

#### (1) 几何学抛物线方程

由于抛物线经过坐标原点，所以抛物线方程为

$$y - h_{\max} = a (x - w)^2$$

将原点坐标  $(0, 0)$  代入方程，得

$$-h_{\max} = aw^2$$

解得

$$a = -\frac{h_{\max}}{w^2}$$

得抛物线方程

$$y = -\frac{h_{\max}}{w^2} (x - w)^2 + h_{\max} \quad (1)$$

展开后得到：

$$y = \frac{2h_{\max}}{w}x - \frac{h_{\max}}{w^2}x^2 \quad (2)$$

式中参数  $h_{\max}$  是未知数，它是由建筑物的高度决定的。从 (1) 式得出：

$$h_{\max} = -\frac{w^2 y}{(x-w)^2 - w^2} \quad (3)$$

将所有建筑物的顶点坐标，存入数组中：

```
struct {
    double x, y;
}height[202];
```

每一个顶点坐标  $(x_i, y_i)$ ，都会有一个高度  $h_i$ ，所有  $h_i$  中的最大值就是参数  $h_{\max}$ ：

$$h_{\max} = -w^2 \max_{1 \leq i \leq 2n} \frac{y_i}{(x_i - w)^2 - w^2} \quad (4)$$

(2) 物理学抛物线方程

设超人跨越时的初始速度为  $v$ ，角度为  $\theta$ ，则有参数方程

$$\begin{aligned} x &= vt \cos \theta \\ y &= vt \sin \theta - \frac{1}{2}gt^2 \end{aligned} \quad (5)$$

式中参数  $t$  是时间（秒）。消除参数  $t$ ，得到抛物线方程

$$y = x \tan \theta - \frac{g}{2v_0^2 \cos^2 \theta} x^2 \quad (6)$$

对比式 (2) 和式 (6) 两个多项式中  $x$  项的系数，我们得到

$$\begin{aligned} \tan \theta &= \frac{2h_{\max}}{w} \\ \frac{h_{\max}}{w^2} &= \frac{g}{2v^2 \cos^2 \theta} \end{aligned} \quad (7)$$

由于  $\sin^2 \theta + \cos^2 \theta = 1$ ，两边同时除以  $\cos^2 \theta$ ，得

$$\begin{aligned} \frac{1}{\cos^2 \theta} &= 1 + \tan^2 \theta \quad (0 < \theta < 90^\circ) \\ \frac{h_{\max}}{w^2} &= \frac{(1 + \tan^2 \theta)g}{2v^2} \end{aligned}$$

解方程 (7)，得到题目的答案：

$$\begin{aligned} \theta &= \arctg\left(\frac{2h_{\max}}{w}\right) \\ v^2 &= 2gh_{\max} + \frac{gw^2}{2h_{\max}} \end{aligned} \quad (8)$$

## 【程序代码】

---

程序名称:	zju1104.c
题    目:	Leaps Tall Buildings
提交语言:	C
运行时间:	0ms
运行内存:	160KB

---

```
#include <stdio.h>
#include <math.h>

#define g 9.8
#define PI 3.1415926535897932384626433832795

//存储所有建筑物的顶点坐标
struct {
    double x, y;
}height[202];

int main() {
    int n;                                //建筑物的数量
    int count,i;                          //count 是顶点坐标的数量
    while (scanf("%d", &n)!=EOF) {
        double dist;
        double width=0;                  //抛物线的总跨度
        count=0;
        //将所有建筑物的顶点坐标都存储到数组中
        for (i=1; i<=n; i++) {
            //建筑物左边的顶点坐标
            count++;
            scanf("%lf%lf", &height[count].y, &dist);
            height[count].x=width;
            //建筑物右边的顶点坐标
            count++;
            height[count].y = height[count-1].y;
            width += dist;
            height[count].x = width;
        }
        width /= 2;                      //抛物线的半跨度
        //根据每一个顶点坐标 ( $x_i, y_i$ ), 计算高度  $h$ , 所有  $h$  中的最大值就是参数  $h_{\max}$ 。
        //根据公式 (4), 计算  $h_{\max}$  (下面的循环中不包含系数 $-W^2$ )
        double h;
```



```

double max = 0;
double wSquare = width*width;
for (i=2; i<=count; i++) {
    if (height[i].y<0.0000001) continue;
    h = height[i].y/((height[i].x-width)*(height[i].x-width)-
        wSquare);
    if (h<max) max=h;
}
//乘上系数-w2
max = -max*wSquare;
//根据公式 (8), 输出超人跨越时的初始角度
printf("%.2lf ", atan(2*max/width)/PI*180);
//根据公式 (8), 输出超人跨越时的初始速度
printf("%.2lf\n", sqrt(2*g*max+wSquare*g/(2*max)));
}
}

```

## ZJU1110-Dick and Jane<sup>[1, 2]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

Dick is 12 years old. When we say this, we mean that it is at least twelve and not yet thirteen years since Dick was born.

Dick and Jane have three pets: Spot the dog, Puff the Cat, and Yertle the Turtle. Spot was  $s$  years old when Puff was born; Puff was  $p$  years old when Yertle was born; Spot was  $y$  years old when Yertle was born. The sum of Spot's age, Puff's age, and Yertle's age equals the sum of Dick's age ( $d$ ) and Jane's age ( $j$ ). How old are Spot, Puff, and Yertle?

Each input line contains four non-negative integers:  $s, p, y, j$ . For each input line, print a line containing three integers: Spot's age, Puff's age, and Yertle's age. Ages are given in years, as described in the first paragraph.

### Sample Input

```

5 5 10 9
5 5 10 10
5 5 11 10

```

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1110>

[2] <http://acm.uva.es/p/v102/10257.html>

## Output for Sample Input

```
12 7 2
13 7 2
13 7 2
```

## Problem Source

University of Waterloo Local Contest 1998.06.06

### 【题目大意】

Dick 12 岁了。当我们这么说时，意思是指从 Dick 出生以来，他至少 12 岁了但未满 13 岁。

Dick 和 Jane 有三只宠物：小狗 Spot，小猫 Puff，和海龟 Yertle。当 Puff 出生时，Spot 已经  $s$  岁；当 Yertle 出生时，Puff 已经  $p$  岁，而 Spot 已经  $y$  岁。Spot，Puff，和 Yertle 的年龄总和等于 Dick 的年龄 ( $d$ ) 和 Jane 的年龄 ( $j$ ) 的总和。Spot，Puff 和 Yertle 多少岁了？

每行输入包含 4 个正整数： $s$ 、 $p$ 、 $y$ 、 $j$ 。

对于每行输入，输出三个整数：Spot 的年龄，Puff 的年龄和 Yertle 的年龄。如第一段所述，年龄用年表示。

### 【算法分析】

本题类似于鸡兔同笼问题，根据已知的数据，推测另外一些未知的数据。

分别设 Spot 的年龄，Puff 的年龄和 Yertle 的年龄为  $S$ 、 $P$  和  $Y$ ， $\Delta s$ ， $\Delta p$  和  $\Delta y$  分别表示 Spot，Puff 和 Yertle 的年龄与年数的差值，取值范围是  $[0, 1]$ 。则

$$S = [S] + \Delta s$$

$$P = [P] + \Delta p$$

$$Y = [Y] + \Delta y$$

式中， $[ ]$  表示取整运算。

根据题意，得到下列方程组

$$S = P + s + \Delta s \quad (1)$$

$$P = Y + p + \Delta p \quad (2)$$

$$S = Y + y + \Delta y \quad (3)$$

$$[S] + [P] + [Y] = d + j \quad (4)$$

式中，因为 Dick 12 岁了，则  $d = 12$ 。

综合式 (1)、式 (2) 和式 (3)，得到

$$y + \Delta y = p + s + \Delta p + \Delta s \quad (5)$$

综合式 (2)、式 (3) 和式 (4)，得到

$$[Y + y + \Delta y] + [Y + p + \Delta p] - [Y] = d + j$$

因为  $y$  和  $p$  是整数，则

$$[Y + \Delta y] + [Y + \Delta p] + [Y] = d + j - y - p \quad (6)$$

令  $\text{temp} = d + j - y - p$

显然  $[Y] = \text{temp} / 3$

(1) 如果  $\text{temp} \% 3 = 0$ , 由式 (6) 可知,  $\Delta y$  和  $\Delta p$  全为 0。

(2) 如果  $\text{temp} \% 3 = 1$ , 由式 (6) 可知,  $\Delta y$  或者  $\Delta p$  中有一个是 1。

如果  $p + s = y$ , 由式 (5) 得

$$\Delta y = \Delta p + \Delta s$$

可以看出 (它们的取值范围是  $[0, 1]$ ), 应该是  $\Delta y$  为 1 (代码中没有  $\Delta y$ , 直接  $y++$ ); 否则是  $\Delta p$  为 1 (代码中没有  $\Delta p$ , 直接  $p++$ )。

(3) 如果  $\text{temp} \% 3 = 2$ , 由式 (6) 可知,  $\Delta y$  和  $\Delta p$  都是 1 (代码中直接  $y++$  和  $p++$ )。

### 【程序代码】

---

程序名称:	zju1110.c
题 目:	Dick and Jane
提交语言:	C
运行时间:	0ms
运行内存:	160KB

---

```
#include <stdio.h>
//代码的功能, 请见上面的算法分析
int main()
{
    int s, p, y, j;
    while(scanf("%d%d%d%d", &s, &p, &y, &j) != EOF)
    {
        int temp = 12 + j - p - y;
        int yAge = temp / 3;
        if (temp % 3 == 1)
        {
            if(s + p == y) y++;
            else p++;
        }
        if (temp % 3 == 2)
        {
            p++;
            y++;
        }
        printf("%d %d %d\n", yAge+y, yAge+p, yAge);
    }
    return 0;
}
```

# ZJU1112-Equidistance<sup>[1, 2, 3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

Alice and Bob haven't met for some time. Bob isn't very happy about this, so he urges Alice to finally make time for a meeting. Let's listen to an extract from a phone call:

*Alice:* ... maybe we should meet on neutral territory.

*Bob:* I've already heard this from you——two years ago.

*Alice:* I know; I just haven't found yet a suitable place that is roughly at the same distance from both yours and mine.

*Bob:* Well, the geometric place of the points that are equidistant from two given points on the surface of a sphere (and the earth is a sphere rather than a disc) is a great circle (namely the one which intersects the great circle through the given points orthogonally at the center of them). If you insist only on approximately equal distances though, we get a zone of some kilometers width and about 40000 km length. Not everything in this zone is water. Thus I think it is a feasible task to find a fitting place.

*Alice:* Now, if I tell you to pick any, we'll certainly land up in Honolulu.

*Bob:* Which is not a too bad idea. So, may I pick any?

*Alice:* As long as I don't have to accept ——but I'm open to suggestions.

*Bob:* Honolulu?

*Alice:* Is it situated on aforementioned geometric place at all?!

*Bob:* Not quite...

Nice. Now let's stop the preliminaries and come to the facts: Given two locations on the earth's surface you can find the geometric place of all equidistant points on the surface. For another given location calculate its distance on the surface to this geometric place. Assume that the earth is a sphere with a radius of 6378 km.

## Input Specification

The input file consists of two parts: a list of locations and a list of queries.

The location list consists of up to 100 lines, one line per location. Each contains a string and two floating-point numbers, separated by whitespace, representing the name of the location, its latitude and its longitude. Names are unique and shorter than 30 characters and do not contain whitespace. Latitudes are between  $-90$  (South Pole) and  $90$  (North Pole) inclusive. Longitudes are between  $-180$  and  $180$  inclusive where negative numbers denote locations west of the meridian and

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1112>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=2412>

[3] <http://acm.uva.es/p/v101/10184.html>

positive numbers denote locations east of the meridian. (The meridian passes through Greenwich, London.) The location list is terminated by a line consisting of a single "#".

Each line in the query list contains three names of locations. You can assume the first location to be Alice's home, the second location to be Bob's home and the third location to be a possible meeting point. The query list is terminated by a line consisting of a single "#".

## Output Specification

For each query, output a line saying "*M* is *x* km off *A/B* equidistance." with *M*, *x*, *A*, *B* appropriately replaced by the location names and the calculated distance rounded to the nearest integer.

If one of the locations in the query didn't occur in the list of locations print "?" instead of the distance.

## Sample Input

```
Ulm          48.700  10.500
Freiburg     47.700   9.500
Philadelphia 39.883 -75.250
SanJose      37.366 -121.933
Atlanta      33      - 84
Eindhoven    52       6
Orlando      28      -82
Vancouver    49      -123
Honolulu     22      -157
NorthPole    90       0
SouthPole    -90      0
#
Ulm Freiburg Philadelphia
SanJose Atlanta Eindhoven
Orlando Vancouver Honolulu
NorthPole SouthPole NorthPole
Ulm SanDiego Orlando
NorthPole SouthPole SouthPole
Ulm Honolulu SouthPole
#
```

## Sample Output

```
Philadelphia is 690 km off Ulm/Freiburg equidistance.
Eindhoven is 3117 km off SanJose/Atlanta equidistance.
Honolulu is 4251 km off Orlando/Vancouver equidistance.
NorthPole is 10019 km off NorthPole/SouthPole equidistance.
Orlando is ? km off Ulm/SanDiego equidistance.
SouthPole is 10019 km off NorthPole/SouthPole equidistance.
SouthPole is 1494 km off Ulm/Honolulu equidistance.
```

## Problem Source

University of Ulm Local Contest 2000

### 【题目大意】

Alice 和 Bob 好久没见面了。Bob 有点不开心了，因此他催促 Alice 找个时间约会。我们来听听他们的一段对话：

Alice: ...也许我们应该在中立的地方会面。

Bob: 这话我两年前都听过了。

Alice: 我知道；只是我现在还没找到一个离你家和我家相同距离的地方。

Bob: 好吧，给定球面上两个坐标点（地球是一个球体并非一个圆盘），则球面上与两个点等距离的几何位置是一条大圆弧（即该圆与给定两点连线的中点垂直相交）。如果你坚持近似等距离的话，那么我们划定一个 40000 公里长、几公里宽的地带。该地带并不全是水域。这样的话，找一个合适的地方是可以实现的。

Alice: 现在，如果让我说要去哪里，我们就去 Honolulu。

Bob: 这个主意还算可以。那，我要挑地方吗？

Alice: 只要我不是被迫接受---但我能听建议。

Bob: Honolulu?

Alice: 它位于刚才说过的几何地理位置上吗？!

Bob: 不完全是...

好吧，我们停止讨论，看看现实：给出地面上两个地点，你可以看到地表上所有到这两点等距离的点构成的几何平面。另外给出一个点，请计算出地表上它到这个几何平面的距离。假设：地球的半径是 6378km。

### 输入格式

输入有二个部份：一系列的位置和一系列的待选地址。

位置列表多达 100 行，每个位置占一行。每行一个字符串和两个浮点数，用空格隔开，表示位置名称及它的纬度和经度。名称是唯一的且不多于 30 个字符（中间无空格）。纬度：-90（南纬）至 90（北纬）。经度：-180（西经）至 180（东经），经度为负表示子午线以西，经度为正表示子午线以东。子午线横跨伦敦的格林威治。位置列表以一个“#”结束。

待选地址列表每行是三个地点名字。你可以假设第一个位置是 Alice 的家，第二个位置是 Bob 的家，第三个位置可能是约会的地方。待选地址列表以一个“#”结束。

### 输出格式

对于每个待选地点，输出一行“*M* is *x* km off *A/B* equidistance.”，这里的 *M*, *A*, *B* 表示位置名称，*x* 表示距离，距离精确到最近的整数。

如果待选地点在位置列表中没有的话，以“?”代替距离。

### 【算法分析】

本题算法来自标程<sup>[1]</sup>。在球表面上，满足与*A*、*B*两点等距的是一个圆，该圆所在的面是

---

[1] <http://www.informatik.uni-ulm.de/acm/Locals/2000/>

一个子午面（即经过球心），线段 $AB$ 就是该面的法线。约会的地点 $M$ 在子午面上，题目要求计算点 $M$ 与线段 $AB$ 的距离。

### （1）纬度和经度

地球的纬度和经度表示如图 9-13 所示。

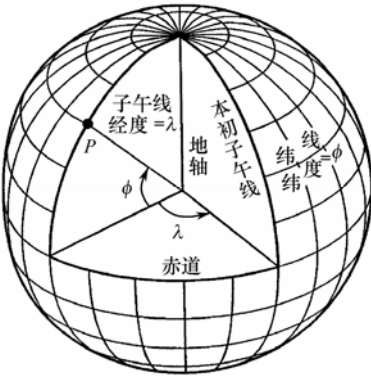


图 9-13 地球的纬度和经度表示

### ① 纬度 (Latitude)

设球面上有一点 $P$ ，通过 $P$ 点作球面的垂线，称之为过 $P$ 点的法线。法线与赤道面的交角，叫做 $P$ 点的地理纬度（简称纬度），通常以字母 $\phi$ 表示。纬度从赤道起算，在赤道上纬度为 $0$ 度，纬线离赤道愈远，纬度愈大，至极点纬度为 $\pm 90$ 度。

### ② 经度 (Longitude)

过 $P$ 点的子午面与通过英国格林威治天文台的子午面所夹的二面角，叫做 $P$ 点的地理经度（简称经度），通常用字母 $\lambda$ 表示。国际规定通过英国格林威治天文台的子午线为本初子午线（或叫首子午线），作为计算经度的起点，该线的经度为 $0$ 度，向东 $0\sim 180$ 度叫东经，向西 $0\sim$

$180$ 度叫西经。

假设地球球心为三维直角坐标系的原点，球心与赤道上 $0$ 经度点的连线为 $x$ 轴，球心与赤道上东经 $90$ 度点的连线为 $y$ 轴，球心与北极点的连线为 $z$ 轴。设地球的半径为 $R$ ，则地面上点的直角坐标与其经纬度的关系为

$$x = R \cos \phi \cos \lambda$$

$$y = R \cos \phi \sin \lambda$$

$$z = R \sin \phi$$

### （2）名称的查找

名称列表：

```
char name[101][64];
```

通过函数 findCity() 实现：

```
int findCity(char *s);
```

对名称 $s$ ，在名称列表 $name$ 中依次比较，返回值是序号，找不到时返回 $-1$ 。

### （3）子午面的法线矢量

地球上所有到 $A$ 、 $B$ 点等距离的点构成的几何平面，是一个子午面，其法线矢量为

$$d_x = x_a - x_b, \quad d_y = y_a - y_b, \quad d_z = z_a - z_b$$

### （4）计算 $M$ 点到该几何平面的距离

有了子午面的法线矢量，就可以把空间问题转换为平面问题，如图 9-14 所示。

图中线段 $MN$ 平行于法线矢量， $MO$ 与 $MN$ 的夹角 $\alpha$ 为

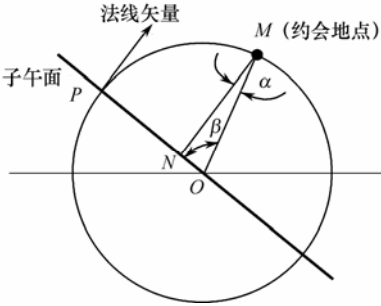


图 9-14 约会地点与子午面之间的位置关系

$$\cos \alpha = \frac{xd_x + yd_y + zd_z}{\sqrt{(d_x)^2 + (d_y)^2 + (d_z)^2}}$$

圆心角 $\beta$

$$\beta = \left| \frac{\pi}{2} - \alpha \right|$$

弧长  $MP$  就等于圆心角 $\beta$ 乘以半径。

### 【程序代码】

---

程序名称:           zju1112.c  
 题     目:           Equidistance  
 提交语言:           C  
 运行时间:           0ms  
 运行内存:           164KB

---

```
#include <stdio.h>
#include <string.h>
#include <math.h>

#define PI 3.14159265358979323846
#define Radius 6378.0

char name[101][64];           //名称列表
int count = 0;                //名称列表的个数

double rad (double x)         //角度转换为弧度
{
    return x * PI / 180.0;
}

//查找名称 s 在列表中的序号
int findCity (char *s)
{
    int i;
    for (i=0; i<count; i++)
        if (strcmp(name[i], s) == 0) return i;    //找到了
    return -1;                                     //没有找到
}

int main ()
{
    count = 0;                                     //位置列表的个数
    double latitude, longitude;                   //纬度和经度
    double x[128], y[128], z[128];               //保存所有位置的直角坐标
```



```

//读取纬度和经度，并转换为直角坐标
while (scanf("%s", name[count]))
{
    if (name[count][0] == '#') break;
    scanf("%lf%lf", &latitude, &longitude);
    x[count] = cos (rad (latitude)) * sin (rad (longitude));
    y[count] = cos (rad (latitude)) * cos (rad (longitude));
    z[count] = sin (rad (latitude));
    ++count;
}
char alice[64], bob[64], meet[64]; //待查询的三个地点
while (scanf("%s", alice))
{
    if (alice[0] == '#') break;
    scanf("%s%s", bob, meet);
    //三个地点在位置列表中的序号
    int a = findCity (alice);
    int b = findCity (bob);
    int m = findCity (meet);
    if (a==-1 || b==-1 || m==-1) //没有找到
    {
        printf("%s is ? km off %s/%s equidistance.\n", meet, alice,
            bob);
        continue;
    }
    //计算子午面的法线矢量
    double dx = x[a] - x[b];
    double dy = y[a] - y[b];
    double dz = z[a] - z[b];
    //计算夹角 $\alpha$ 
    double angle = acos ((x[m] * dx + y[m] * dy + z[m] * dz)
        / sqrt (dx*dx + dy*dy + dz*dz));
    //计算约会地点到子午面的弧长
    double dist = Radius * fabs (0.5 * PI - angle);
    //约会地点在子午面上
    if (dx == 0.0 && dy == 0.0 && dz == 0.0)
        dist = 0;
    printf("%s is %d km off %s/%s equidistance.\n",
        meet, (int) floor(dist+0.5), alice, bob);
}
return 0;
}

```

# ZJU1114-Problem Bee<sup>[1, 2, 3]</sup>

Time Limit: 1 Second

Memory Limit: 32768KB

Special Judge

## Background

Imagine a perfectly formed honeycomb, spanning the infinite Cartesian plane. It is an interlocking grid composed of congruent equilateral hexagons. One hexagon is located so that its center is at the origin and has two corners on the X-axis. A bee must be very careful about how it travels in order not to get lost in the infinite plane. To get from an arbitrary point  $A$  to another arbitrary point  $B$ , it will first head from  $A$  to the exact center of the hexagon in which  $A$  is located. Then, it will travel in a straight line to the exact center of an adjacent hexagon. It will move from center to adjacent center until it has reached the hexagon containing point  $B$ . At the destination hexagon, it will move from the center to point  $B$ . In all cases the bee will take a path of minimal distance that obeys the rules. The figure below demonstrates one possible minimal path from point  $A$  to point  $B$ .

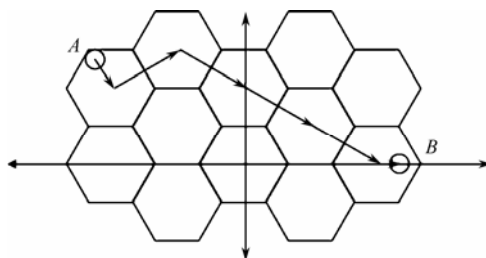


Figure 9-15

## Input

Input will be in the form of 5 floating point numbers per line. The first number will be the length, in centimeters, of the sides of the hexagons. The next two numbers will be the  $x$  and  $y$  coordinates of point  $A$ , followed by the  $x$  and  $y$  coordinates for point  $B$ . The input will be terminated by a line containing five zeroes. Neither point  $A$  nor point  $B$  will ever be exactly on a border between hexagons.

## Output

For each line of the input, output the minimum length of a path from point  $A$  to point  $B$ , in centimeters, to the nearest .001 centimeters.

Example

## Input

1.0 -3.2 2.2 3.3 0

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1114>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1518>

[3] <http://www.acmgnyr.org/year2000/>

```

9 1 4 5 1
0.1 .09 0 .21 0
0 0 0 0 0

```

## Output

```

7.737
5.000
0.526

```

## Problem Source

Greater New York 2000

### 【题目大意】

#### 背景

想象一个巧夺天工的蜂巢，占据了整个直角平面。它们是全等六边形，而且一个紧挨着一个。有一个六边形的中心位于原点，其两个角点位于  $x$  坐标上。为了在这么一个广阔的蜂巢里不迷失方向，飞行的蜜蜂必须格外谨慎。从任意  $A$  点到任意  $B$  点，蜜蜂首先进入  $A$  所在六边形的中心。然后，它就可以沿直线飞向邻近的六边形。小蜜蜂就这样从一个六边形的中心飞向另一个邻接的六边形中心，一直飞到  $B$  点所在的六边形。接着，它就直接从中心飞向  $B$  点。在任何情况下，蜜蜂总是根据规则选择最小距离的路线。图 9-15 是蜜蜂从  $A$  点飞向  $B$  点的一种可能的最短路径。

#### 输入格式

每行输入 5 个浮点数。第一个数是六边形一边的长度（厘米），然后是  $A$  点坐标  $x$ ,  $y$  和  $B$  点坐标  $x$ ,  $y$ 。当一行是 5 个 0 时，输入结束。 $A$  点或  $B$  点不会出现在两个邻接六边形的公共边上。

#### 输出格式

对于每行输入，输出蜜蜂从  $A$  点飞向  $B$  点的最短路径，精确到小数点三位。

### 【算法分析】

本题算法参考自Gabriel Nivasch的个人网页<sup>[1]</sup>。为了便于计算蜜蜂沿六边形的中心飞行，建立一个以六边形中心和六边形水平边中点为网格的坐标系，如图 9-16 所示：

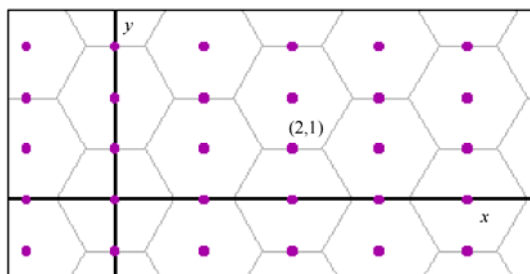


图 9-16 与直角坐标系重叠的网格坐标系

标记网格坐标系的点为  $(K, L)$ ,  $K$  和  $L$  为整数。从图 9-16 中看出，位于六边形中心的点， $K$  和  $L$  具有相同的奇偶性。

[1] [http://yucs.org/~gnivasch/ACM\\_B/index.html](http://yucs.org/~gnivasch/ACM_B/index.html)

### (1) 直角坐标和网格坐标的转换

设六边形的边长为  $side$ ，则直角坐标  $(x, y)$  和网格坐标  $(K, L)$  的转换关系为

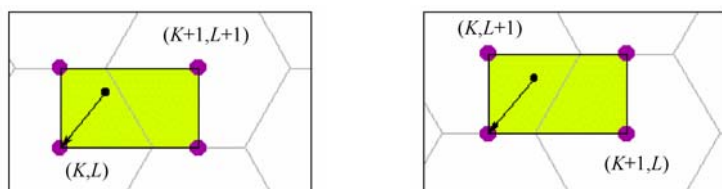
$$x = \frac{3}{2}K side, \quad y = \frac{\sqrt{3}}{2}L side \quad (1)$$

则直角坐标  $(x, y)$  的网格坐标  $(K, L)$  为

$$K = \frac{2x}{3side}, \quad L = \frac{2y}{\sqrt{3} side} \quad (2)$$

### (2) 计算 $A$ 、 $B$ 两点的网格坐标

式 (2) 在计算时，只取整数部分，这样就得到了直角坐标  $(x, y)$  左下角的网格坐标  $(K, L)$ ，如图 9-17 所示：



(a)  $(K, L)$  在六边形中心

(b)  $(K, L)$  在六边形边线的中点

图 9-17 网格坐标的计算

如果  $(K, L)$  的奇偶性相同，它就在六边形中心，否则在六边形边线的中点上。这时分为两种情况：

#### (a) $(K, L)$ 在六边形中心。

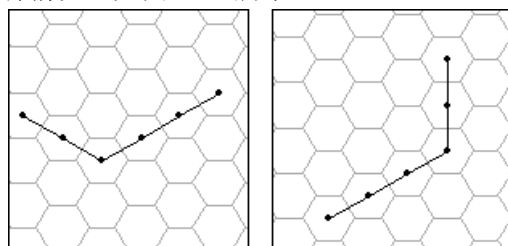
由于直角坐标  $(x, y)$  也可能在六边形  $(K+1, L+1)$  中，就需要判断一下。计算  $(x, y)$  与两个六边形中心的距离，取距离小的那个六边形。

#### (b) $(K, L)$ 在六边形边线的中点。

这时  $(K, L)$  的值应该修正为  $(K, L+1)$  或者  $(K+1, L)$ 。也是计算  $(x, y)$  与两个六边形中心的距离，取距离小的那个六边形。

### (3) 计算蜜蜂飞行的距离

蜜蜂飞行的路径有两种情况，如图 9-18 所示：



(a) 全部为斜角飞行

(b) 斜角和直线飞行

图 9-18 蜜蜂的飞行路径

蜜蜂飞行一个六边形，其飞行路径的长度为  $\sqrt{3} side$ 。根据  $|L_B - L_A|$  和  $|K_B - K_A|$  来判别：

①若  $|L_B - L_A| < |K_B - K_A|$ ，属于第一种情况；

②若  $|L_B - L_A| > |K_B - K_A|$ ，属于第二种情况。

分段计算路径的长度。当然还要考虑  $A$ 、 $B$  两点就在同一个六边形中的情况。

## 【程序代码】

---

程序名称:	zju1114.c
题    目:	Problem Bee
提交语言:	C
运行时间:	0ms
运行内存:	160KB

---

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define Root 0.8660254                //  $\sqrt{3}/2$ 
//计算平方
double square(double x) {
    return x*x;
}

//计算六边形内, 坐标 (x,y) 与中心点 (k,l) 之间的距离, side 是边长
double dis(int k, int l, double x, double y, double side)
{
    return sqrt(square(1.5*side*k-x) + square(Root*side*l-y));
}

int main()
{
    double side, xa, ya, xb, yb;      //六边形的边长, A 和 B 点的坐标
    int ka, la;                       //A 点的网格坐标
    int kb, lb;                       //B 的网格坐标
    while(scanf("%lf%lf%lf%lf%lf", &side, &xa, &ya, &xb, &yb) && (side || xa || ya || xb || yb))
    {
        //计算 A、B 两点的网格坐标
        ka = floor(2*xa/3/side);
        la = floor(2*ya/sqrt(3)/side);
        kb = floor(2*xb/3/side);
        lb = floor(2*yb/sqrt(3)/side);
        //A 点网格坐标调整
        //(ka, la)在六边形的边线上
        if(ka%2!=la%2)
        {
            if( dis(ka, la+1, xa, ya, side)>dis(ka+1, la, xa, ya, side)) ka++;
            else la++;
        }
        //(ka, la)在六边形的中心
        else if(dis(ka+1, la+1, xa, ya, side)<dis(ka, la, xa, ya, side))
        {
```

```

        ka++; la++;
    }
    //B 点网格坐标调整
    if (kb%2!=lb%2)
    {
        if(dis(kb,lb+1,xb,yb,side)>dis(kb+1,lb,xb,yb,side)) kb++;
        else lb++;
    }
    else if(dis(kb+1,lb+1,xb,yb,side)<dis(kb,lb,xb,yb,side))
    {
        kb++; lb++;
    }
    //计算 A 和 B 两点与网格中心之间的距离 ( $\Delta A$  和  $\Delta B$ )
    double dA = dis(ka,la,xa,ya,side);
    double dB = dis(kb,lb,xb,yb,side);
    //计算蜜蜂飞越的距离
    double length;
    //A 和 B 两点在同一个六边形内
    if (ka==kb && la==lb)
        length = sqrt(square(xa-xb) + square(ya-yb));
    else if(fabs(ka-kb)>fabs(la-lb))
        length = fabs(ka-kb) * sqrt(3)*side + dA +dB;
    else
        length = sqrt(3)*side*(fabs(kb-ka)+ (fabs(lb-la)-fabs(kb-ka))
        /2) + dA +dB;
    printf("%.3lf\n",length);
}
return 0;
}

```

## ZJU1123-Triangle Encapsulation<sup>[1、2、3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

Each line of the input to your program will specify the  $x$ - and  $y$ -coordinates of the three vertices of a triangle in the  $xy$  -plane, starting at some vertex and proceeding clockwise. For each vertex, the  $x$ -coordinate will be specified before the  $y$ -coordinate. Each  $x$ - and  $y$ -coordinate will be an integer in the range  $-9\cdots 9$ . The input file will consist of one or more lines.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1123>

[2] <http://acm.tju.edu.cn/toj/showp2096.html>

[3] <http://acm.uva.es/archive/nuevoportal/data/problem.php?p=2078>

For each triangle specified in the input, the output will list all points in the  $xy$ -plane which have integer coordinates and lie in the triangle's interior. (Such points are said to be encapsulated by the triangle.) Points that lie on one of the triangle's borders (in particular, the vertices of the triangle) will not be listed in the output.

Each triangle specified in the input will encapsulate at least one point with integer coordinates.

Within each line of input, the difference between the largest and smallest  $x$ -coordinate will not exceed 9.

### Sample Input

```
4 4 2 -1 -2 2
2 4 4 -3 0 -1
```

### Sample Output

```
Program 4 by team X
                                (2, 3) (3, 3)
(-1, 2) (0, 2) (1, 2) (2, 2) (3, 2)
                                (0, 1) (1, 1) (2, 1)
                                    (1, 0) ( 2, 0)
                                (2, 3)
                                (2, 2)
(1, 1) (2, 1)
(1, 0) (2, 0) (3, 0)
(1, -1) (2, -1) (3, -1)
                                (3, -2)
End of program 4 by team X
```

Figure 1 illustrates the first triangle of the above example, and the points with integer coordinates in its interior. Note that  $(1, 3)$  lies on this triangle's border and is therefore not encapsulated by the triangle.

Points  $(x, y)$  which are listed on one line of output have the same  $y$ -coordinate; the  $x$ -coordinates of these points are in increasing order from left to right.

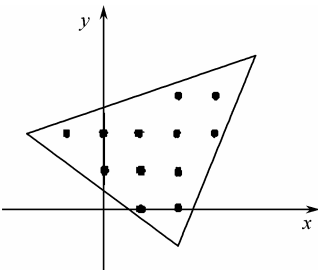


Figure 9-19

The  $y$ -coordinates of points in the interior of a triangle will be in decreasing order from one line of output to the next line.

Each  $(x, y)$  point will be printed in a nine-column output field. Points that have the same  $x$ -coordinate (and pertain to the same triangle) will be listed in the same nine-column output field. Here is a formatting template between two lines of the above output:

```
                                (2, 3) (3, 3)
12345678901234567890123456789012345678901234567890
(-1, 2) (0, 2) (1, 2) (2, 2) (3, 2)
```

Each nine-column output field has the following layout (unless it is blank):

Column 1: opening parenthesis

Columns 2-3 : x coordinate (right justified)

Column 4: comma

Columns 5-7 : y -coordinate (right justified)

Column 8 : closing parenthesis.

Column 9: blank space to separate current nine-column field from the next one, if there is a next one.

The line(s) containing the smallest  $x$ -value pertaining to a particular triangle (such as -1 for the first triangle in the above example) will not start with a blank output field.

There will not be a blank output field between two non-blank ones on a given line.

After listing the points encapsulated by each triangle, your program will produce one blank line of output. There will not be any additional blank lines in the output.

## Problem Source

Rocky Mountain 2000

### 【题目大意】

输入中的每一行，是  $x$ - $y$  坐标系中三角形三个顶点的坐标，从一个顶点开始，按顺时针方向。每个顶点的  $x$  坐标在  $y$  坐标之前。 $x$ - $y$  坐标的范围是：-9~9。输入有一行或多行。

对于输入的每个三角形，输出该三角形内的整数坐标（即被三角形封闭的点）。但不包括其边上的坐标（特别是顶点坐标）。

输入的三角形内部至少有一个整数坐标。

对每行输入， $x$  坐标的最大值和最小值之差不超过 9。

图 9-19 就是输入样例的第一个三角形，其内部的点都是整数坐标。注意：(1, 3) 位于三角形的边上，因此该坐标没有被三角形封闭。

$y$  坐标相同的点 ( $x, y$ )，输出在同一行；而这些点的  $x$  坐标从左到右升序输出。

$y$  坐标降序输出，数值最大的输出在第一行，然后第二行等。

每个坐标宽为九列，称为一个输出域。相同  $x$  值的坐标，输出域应对齐。

.....

如果是  $x$  坐标最小值所在的行（如样例数据第一个三角形中的-1），该行的第一个输出域不为空。

在一行中，两个非空的输出域之间，没有空的输出域。

每个三角形的后面有一个空行，其他地方没有空行。

### 【算法分析】

本题要求输出位于三角形内部的整数坐标点。算法参考自博客“暑假训练之记录”<sup>[1]</sup>。

#### (1) 计算包围三角形的矩形

由于三角形特别小（ $x$ - $y$  坐标的范围是-9~9），使用一个平行于坐标轴的矩形把三角形套起来。显然矩形的边界是三角形三个顶点坐标中的最大、最小值，如图 9-20 所示。

---

[1] <http://www.cppblog.com/bnugong/archive/2008/07/22/56872.html>



## (2) 判断三角形内部的点

用数组 `map` 表示矩形内的所有整数坐标点：

```
char map[50][50];
```

如果点  $(x, y)$  在三角形内，值为 1，否则为 0。

采用计算三角形面积的办法判断点是否在三角形内部，如图 9-21 所示。

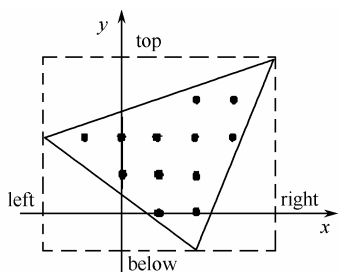
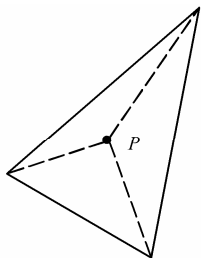
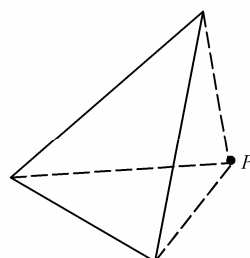


图 9-20 包围三角形的矩形



(a) 点在三角形内部



(b) 点在三角形外部

图 9-21 点在三角形内外的判断

如果点  $P$  在三角形内部，该点与三角形三个顶点构成的三个三角形面积的和等于大三角形的面积，否则不相等。如果点  $P$  在三角形的边界上，则三个三角形中，必然有一个面积为 0。

但是  $x$ - $y$  坐标的范围是  $-9 \sim 9$ ，直接用  $xy$  当作数组 `map` 的下标，负数值是非法的，所以在坐标上加 10 即可，输出时再减去 10。

## (3) 三角形面积的计算

在计算面积时，采用了叉乘的办法，如图 9-22 所示。

由函数 `area()` 实现三角形面积的计算：

```
int area(int a,int b,int c,int d);
```

面积公式。为  $s = \frac{1}{2} \begin{vmatrix} a & b \\ c & d \end{vmatrix}$ ，如果  $s < 0$ ，则求绝对值。

## (4) 计算包围三角形内整数点的矩形

根据数组 `map` 的值，就可以判断要输出的整数点区域，它也是一个矩形，如图 9-23 所示：

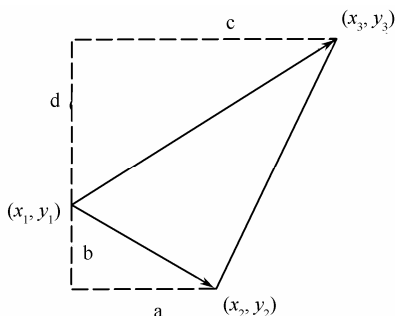


图 9-22 三角形面积的计算

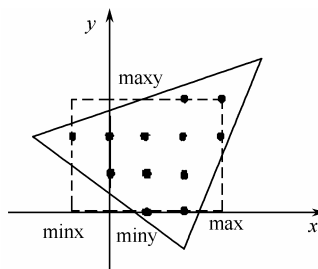


图 9-23 包围三角形内部整数点的矩形

在矩形区域内的点  $(i, j)$  ( $\text{minx} \leq i \leq \text{maxx}$ ,  $\text{miny} \leq j \leq \text{maxy}$ )，如果数组 `map` 的值为 1，则输出该坐标，否则为空输出域。使用变量 `border` 控制每行的右边界，以免在行末输出空输出域。

## 【程序代码】

---

程序名称:	zju1123.c
题    目:	Triangle Encapsulation
提交语言:	C
运行时间:	0ms
运行内存:	160KB

---

```
#include<stdio.h>
#include<string.h>
//计算最小值
int min(int a, int b, int c) {
    int temp = (a<b)?a:b;
    return (temp<c)?temp:c;
}
//计算最大值
int max(int a, int b, int c) {
    int temp = (a>b)?a:b;
    return (temp>c)?temp:c;
}
//计算面积, 叉乘的办法, 是实际面积的两倍
int area(int a,int b,int c,int d)
{
    int s = a*d-b*c;
    if (s<0) s = -s;
    return s;
}

int main()
{
    int x1,x2,x3,y1,y2,y3;           //三角形三个顶点的坐标
    int right,top,left,below;        //套住三角形的矩形边界
    int minx,miny,maxx,maxy;        //套住三角形内部整数点的矩形边界
    int ss,s1,s2,s3;                //三角形的面积
    char map[50][50];                //保存坐标点在三角形内外的状态
    int border;                      //控制行末的空输出域
    int i,j;
    printf("Program 4 by team X\n");
    while(scanf("%d%d%d%d%d", &x1,&y1,&x2,&y2,&x3,&y3)!=EOF)
    {
        minx = miny = 10;
        maxx = maxy = -10;
        ss = area(x2-x1,y2-y1,x3-x1,y3-y1);
        memset(map, 0, sizeof(map));
```

```

//计算包围三角形的矩形
right = max(x1, x2, x3);
top   = max(y1, y2, y3);
left  = min(x1, x2, x3);
below = min(y1, y2, y3);
//判断该矩形内哪些点是三角形内部的整数点
for(i=left; i<=right; i++)
    for(j=below; j<=top; j++)
    {
        s1 = area(x1-i,y1-j,x2-i,y2-j);
        s2 = area(x1-i,y1-j,x3-i,y3-j);
        s3 = area(x2-i,y2-j,x3-i,y3-j);
        //若某个面积为 0, 表示该点在边界上
        if(s1 && s2 && s3 && (s1+s2+s3)==ss)
        {
            map[i+10][j+10] = 1;
            //计算包围三角形内整数点的矩形
            if(i+10<minx) minx = i+10;
            if(i+10>maxx) maxx = i+10;
            if(j+10>maxy) maxy = j+10;
            if(j+10<miny) miny = j+10;
        }
    }
for(j=maxy; j>=miny; j--)
{
    //计算行末的右边界
    for(i=minx; i<=maxx; i++)
        if(map[i][j]) border = i;
    //输出三角形内整数点的坐标
    for(i=minx; i<=border; i++)
    {
        if (i!=minx) printf(" ");
        if (map[i][j]) printf("(%2d, %2d)",i-10,j-10);
        else printf("          ");
    }
    printf("\n");
}
printf("\n");

}

printf("End of program 4 by team X\n");
return 0;
}

```



In the normalized binary scientific notation there is only one binary digit preceding the binary point, and it is always 1 (except if the floating point number is the number zero). It is called the characteristic. Since it is always 1, it does not explicitly appear in the 16-bit representation of Figure 9-25.

Since the number 1.01001100 is in binary, the digits to the right of the binary point represent successive negative powers of 2. Therefore, the number shown in Figure 2 can be re-written as

$$-(1 + 2^{-2} + 2^{-5} + 2^{-6}) \times 2^3 = -10.375 = -1.0375 \times 10^1$$

Here the result has been expressed using scientific notation (base ten). Most programming languages would output this result as  $-1.0375e+001$ . The use of a lower case "e" or upper case "E", and the number of leading zeros in the exponent can vary from one programming language to another.

An important exception: if all bits except the sign bit are zero, then the floating point number is zero, regardless of the sign bit.

## Sample Input

```
1100001001001100
0011111100000000
1011111110000000
0000000010101010
0011011111100000
1001111011100000
0101011001010101
0100011011101101
0111111111111111
1100001000101100
0000000000000000
1000000000000000
```

## Sample Output

```
Program 6 by team X
-1.037500e+001
 1.000000e+000
-1.500000e+000
 1.804180e-019
 7.324219e-003
-2.182787e-010
 1.117389e+007
 2.465000e+002
 3.682143e+019
-9.375000e+000
 0.000000e+000
 0.000000e+000
End of program 6 by team X
```

The input contains an undetermined number of lines. Each line contains, starting in column 1, exactly 16 printable characters which can be any permutation of 0s and 1s. The first character represents the sign bit, the next seven characters, the exponent, and the last eight characters, the mantissa of a floating point number in normalized binary scientific notation according to the conventions discussed on the preceding page.

Each floating point number read from the input will be written by your program to the output file, one number per line.

The following formatting conventions are required in your Output:

The floating point numbers will be displayed in scientific notation (base ten).

Column 1 will either be blank (indicating a positive number or zero) or contain a minus sign (indicating a negative number).

Column 2 will contain a digit.

Column 3 will contain the decimal point.

Columns 4~9 will contain six additional significant digits of the floating point number.

The remaining columns will specify the exponent of scientific notation. The sign and magnitude of the exponent must of course be correct, but the exact format in which the exponent is displayed will be determined by the programming language you use.

## Problem Source

Rocky Mountain 2000

### 【题目大意】

大多数计算机系统的硬件层中，都用浮点数表示二进制科学计数法。我们假设某台计算机用 16 位字存储浮点数，如图 9-24 所示。在通常的 Intel 芯片里，指数 8 位，尾数有 23 位。除此之外，表示方法如同图 9-24 所示。

样例中表示的十进制数是-10.375。

如果最左端（符号位）是 1，那么该数是负数，否则是正数。

接下来的 7 位（本例中的 1000010）表示二进制科学计数法的指数部分。下面的运算得出实际的指数。先计算二进制数 1000010：

$$1 \times 64 + 0 \times 32 + 0 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1 = 66$$

66 减去偏差修正 63，得到数字 3，这个数据下面将要用到。

（偏差修正允许负数的指数。只要指数有 7 位（如本例所示），那么偏差修正就是  $63 = 2^{(7-1)} - 1$ 。

最后 8 位（本例中 01001100）构成尾数（十进制小数部分的二进制科学计数法表示）。下面将看到如何使用尾数。

符号位，尾数，指数都是用来计算浮点数的值：……

在尾数之前的小数点是二进制的小数点，与十进制中的意思是一样的。

在二进制科学计数法中，小数点前只有一个二进制数，这个数通常是 1（除非该浮点数是 0），称为特征位。因为它总是 1，所以在图 9-23 的 16 位中没出现。

1.01001100 是一个二进制数。该数小数点的右边是连续的 2 的负指数。因此图 9-25 中的

数又可以写成:

$$-(1 + 2^{-2} + 2^{-5} + 2^{-6}) \times 2^3 = -10.375 = -1.0375 \times 10^1$$

其结果是用十进制表示的科学计数法。一般的编程语言都输出该数为 $-1.0375\text{e}+001$ 。可以用小写“e”或大写“E”。指数中的前缀0可以随着编程语言而变化。

例外: 如果除了符号位外其它位都是0, 那么就忽略符号位, 该数为0。

输入的行数不确定。每行从第1列开始, 16个字符, 是0和1的一个任意排列。第一个字符是符号位, 接着的7位是指数, 最后8位是二进制科学计数法表示的浮点数尾数。

对应每个输入的浮点数输出相应的结果, 每行一个数。

以下是输出格式要求:

浮点数以十进制的科学计数法输出。

每行的第一列是一个空行(表示正数或者0), 或者是一个减号(表示负数)。

第二列是一个十进制数。

第三列是一个小数点。

第4~9列是该浮点数剩余的6个有效数字。

剩下的几列是用来表示科学计数法的指数。指数的符号和指数大小应该正确。但具体的指数格式取决于你的编程语言。

### 【算法分析】

一个类似于IEEE存储方式的浮点数, 转换为规格化的十进制数字。

#### (1) 数据结构

题目中输入的数据, 用字符串表示为

```
char binary[20];
```

将输入的数据 binary 转换成二进制数字为

```
unsigned short floating;
```

从 floating 中分离出来的二进制数字指数部分为

```
unsigned char high;
```

规格化十进制数字的指数(exponent)部分为

```
int expo;
```

从 floating 中分离出来的二进制数字尾数部分(已经包含2的位权):

```
unsigned char low;
```

规格化十进制数字的尾数(mantissa)部分为

```
double mant;
```

#### (2) 计算公式

一个IEEE格式(二进制科学计数法)的数据, 其十进制数 real 为

$$\text{real} = \left(1 + \frac{\text{low}}{256}\right) \times 2^{\text{high}-63}$$

本题中 high 只有低7位是有效的。由于 low 已经包含2的位权, 直接除以256就是十进制数字。

规格化十进制数字的指数:

$$\text{expo} = \text{floor}\left(\frac{\ln \text{real}}{\ln 10}\right)$$

其中 `floor(double x)` 是 C 语言函数，返回不大于参数  $x$  的最大整数。  
规格化十进制数字的尾数为

$$\text{mant} = \frac{\text{real}}{10^{\text{expo}}}$$

## 【程序代码】

---

程序名称:           zjul125.c  
 题    目:           Floating Point Numbers  
 提交语言:           C  
 运行时间:           0ms  
 运行内存:           160KB

---

```
#include <stdio.h>
#include <math.h>

#define LN10 2.30258509299405           //ln10

void Solved(char binary[])
{
    int i;
    unsigned char low, high;             //二进制数据的尾数和指数
    int expo;                            //规格化十进制数字的指数部分
    double mant;                         //规格化十进制数字的尾数部分
    unsigned short floating;             //将 binary 转换成二进制数字
    //将字符型的数据 binary 转换成二进制数字
    floating = 0;
    for(i = 0; i < 16; i++)
    {
        floating = (floating<<1);
        if(binary[i] == '1') floating++;
    }
    //分离数据，得到二进制数据的尾数和指数部分
    low = floating & 0x00FF;
    high = floating>>8;
    //数字'0'
    if(floating == 0x8000 || floating == 0)
        printf(" 0.000000e+000\n");
    else
    {
        if((floating & 0x8000) != 0) printf("-");
        else printf(" ");
        //计算规格化十进制数字的尾数和指数
        mant = (1+low/256.0)*pow(2.0, (high&0x7F)-63);
    }
}
```



```

        expo = floor(log(mant)/LN10);
        mant = mant/pow(10.0, expo);
        printf("%.6lfe", mant);                //输出尾数
        //输出指数部分
        if(expo>=0) printf("+");
        else
        {
            printf("-");                        //处理负号
            expo = -expo;
        }
        printf("%03d\n", expo);
    }
}

int main()
{
    char binary[20];
    printf("Program 6 by team X\n");
    while (scanf("%s", binary)!=EOF)
        Solved(binary);
    printf("End of program 6 by team X\n");
    return 0;
}

```

## ZJU1128-Atlantis<sup>[1、2、3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

There are several ancient Greek texts that contain descriptions of the fabled island Atlantis. Some of these texts even include maps of parts of the island. But unfortunately, these maps describe different regions of Atlantis. Your friend Bill has to know the total area for which maps exist. You (unwisely) volunteered to write a program that calculates this quantity.

### Input

The input file consists of several test cases. Each test case starts with a line containing a single integer  $n$  ( $1 \leq n \leq 100$ ) of available maps. The  $n$  following lines describe one map each. Each of these lines contains four numbers  $x_1$ ;  $y_1$ ;  $x_2$ ;  $y_2$  ( $0 \leq x_1 < x_2 \leq 100000; 0 \leq y_1 < y_2 \leq 100000$ ), not necessarily integers. The values  $(x_1; y_1)$  and  $(x_2; y_2)$  are the coordinates of the top-left resp. bottom-right corner of the mapped area.

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1128>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1151>

[3] <http://acm.uva.es/archive/nuevoportal/data/problem.php?p=2184>

The input file is terminated by a line containing a single 0. Don't process it.

## Output

For each test case, your program should output one section. The first line of each section must be "Test case #k", where k is the number of the test case (starting with 1). The second one must be "Total explored area: a", where a is the total explored area (i.e. the area of the union of all rectangles in this test case), printed exact to two digits to the right of the decimal point.

Output a blank line after each test case.

## Sample Input

```
2
10 10 20 20
15 15 25 25.5
0
```

## Sample Output

```
Test case #1
Total explored area: 180.00
```

## Problem Source

Mid-Central European Regional Contest 2000

### 【题目大意】

有一些古希腊著作描述传说中的亚特兰蒂斯 (Atlantis) 岛，其中有一些著作还有部分岛屿的插图。但不幸的是，这些图是 Atlantis 岛的不同区域。你的朋友 Bill 想知道，从图上看该岛的面积到底有多大。你愿意写一个程序计算面积的大小。

### 输入格式

输入有多组测试数据。每组数据的第一行是一个整数  $n$  ( $1 \leq n \leq 100$ )，表示有用的地图数量。接下来  $n$  行，每行描述一幅地图。每行是四个数， $x_1; y_1; x_2; y_2$  ( $0 \leq x_1 < x_2 \leq 100000$ ,  $0 \leq y_1 < y_2 \leq 100000$ )。( $x_1; y_1$ )和( $x_2; y_2$ )分别是地图的左上角，右下角的坐标。

当一行是 0 时，输入结束，也不需处理。

### 输出格式

对每组测试数据，程序输出一栏。该栏的第一行是 "Test case #k"， $k$  是测试例的编号 (从 1 开始)。第二行是 "Total explored area: a"， $a$  是计算的总面积 (即该测试例中，所有矩形面积的并集)，精确到小数点后两位。

在每组测试数据之后空一行。

### 【算法分析】

在平面上有很多矩形，矩形的两边与坐标轴平行，矩形有可能相互重叠，计算矩形在平面上占用的面积和 (即重叠部分只计算一次)。

作所有矩形的包络矩形 (即包围所有矩形的最小矩形)，延伸所有矩形的两边与包络矩形相交，构成一个网格。用逻辑值表示网格被矩形覆盖的情况，如果某一个网格被矩形覆盖，逻辑

辑值为 1，否则为 0。统计所有逻辑值为 1 的网格面积即可。

### (1) 数据结构

样例数据比较简单，补充一个样例：

3

10 25 55 85

40 10 100 65

90 50 135 85

0

这是 3 个矩形，按前面介绍的方法建立网格，如图 9-26 所示：

矩形的坐标为

```
struct {
    double x1, y1;
    double x2, y2;
}rect[102];
```

所有网格的坐标为

```
double x[202],y[202];
```

网格被矩形覆盖的情况（用逻辑值 0/1 表示）为

```
char valid[202][202];
```

矩形的数量为

```
int n;
```

### (2) 构建网格

为了构建网格，需要对坐标排序。如  $x$  坐标的排序为

```
qsort(x, 2*n, sizeof(double), cmp);
```

其中参数 **cmp** 是排序因子，由函数实现：

```
int cmp(const void *a,const void *b)
```

在该函数内，指定按坐标升序排序。

### (3) 计算网格被矩形覆盖的情况

由于成功地构建了网格，只要计算每个矩形覆盖网格的情况。对矩形  $i$  ( $0 \leq i < n$ )，找出其左上角 (**left, top**) 和右下角 (**right, bottom**) 的网格坐标（不是直角坐标），将该区域内的网格全部置 1。

### (4) 统计面积

对每一个网格，如果其逻辑值为 1，表示该网格被矩形覆盖，对应的矩形面积就要计算在总面积内。统计所有网格对应的矩形面积，就是所求的答案。

## 【程序代码】

程序名称：	zju1128.c
题 目：	Atlantis
提交语言：	C
运行时间：	0ms
运行内存：	204KB

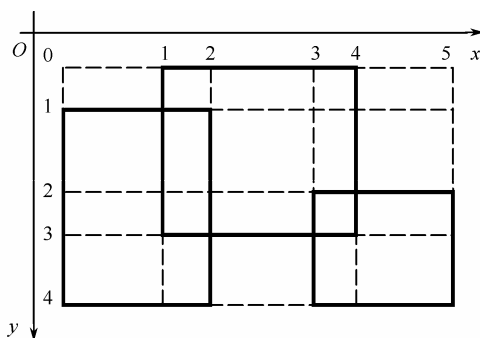


图 9-26 计算矩形覆盖的网格

```

#include<stdio.h>
#include<stdlib.h>
#include<memory.h>
#include<math.h>

struct {
    double x1, y1;
    double x2, y2;
}rect[102];           //所有矩形的坐标
double x[202],y[202]; //网格的坐标
char valid[202][202]; //网格被矩形覆盖的情况
int n;               //矩形的数量

//排序因子，按坐标升序
int cmp(const void *a,const void *b)
{
    double temp = *(double *)a-*(double *)b;
    if (fabs(temp)<1e-5) return 0;
    else if (temp>0) return 1;
    else return -1;
}

//计算网格被矩形覆盖的情况
void solve(){
    int i, j, il, jl;
    int left,top,right,bottom;
    //对每一个矩形，将其覆盖的网格单元全部置 1
    for(i = 0 ; i < n; i++) {
        //找到第 i 个矩形左边的坐标序号
        j = 0 ;
        while (fabs(x[j]-rect[i].x1)>0 && j < 2*n) j++;
        left = j;
        //找到第 i 个矩形顶部的坐标序号
        j = 0;
        while (fabs(y[j]-rect[i].y1)>0 && j < 2*n) j++;
        top = j;
        //找到第 i 个矩形右边的坐标序号
        j = 0 ;
        while (fabs(x[j]-rect[i].x2)>0 && j < 2*n) j++;
        right= j;
        //找到第 i 个矩形底部的坐标序号
        j = 0;
        while (fabs(y[j]-rect[i].y2)>0 && j < 2*n) j++;
    }
}

```

```

        bottom = j;
        //将该矩形所覆盖的网格，全部置 1
        for(il = left ; il < right ; il++)
            for(jl = top ; jl < bottom ; jl++)
                valid[il][jl] = 1;
    }
}

int main()
{
    int i, j;
    int iCase=1;
    while(scanf("%d",&n) && n)
    {
        //读取矩形的数据，并构建网格坐标
        j = 0;
        for(i = 0 ; i < n ; i++)
        {
            scanf("%lf%lf%lf%lf", &rect[i].x1, &rect[i].y1, &rect[i].x2,
                &rect[i].y2);
            x[j] = rect[i].x1;
            y[j] = rect[i].y1;
            j++;
            x[j] = rect[i].x2;
            y[j] = rect[i].y2;
            j++;
        }
        //网格坐标排序
        qsort(x, 2*n, sizeof(double), cmp);
        qsort(y, 2*n, sizeof(double), cmp);
        memset(valid, 0, sizeof(valid));
        //判断网格被矩形覆盖的情况
        solve();
        //统计面积，累加被矩形覆盖的网格面积
        double sum = 0;
        for(i = 0; i < 2*n; i++)
            for(j = 0; j < 2*n; j++)
                sum += valid[i][j]*(x[i+1] - x[i])*( y[j+1] - y[j]);
        printf("Test case #%d\n", iCase++);
        printf("Total explored area: %.2f\n\n", sum);
    }
    return 0;
}

```

# ZJU1133-Smith Numbers<sup>[1, 2, 3]</sup>

---

Time Limit: 1 Second

Memory Limit: 32768KB

---

While skimming his phone directory in 1982, Albert Wilansky, a mathematician of Lehigh University, noticed that the telephone number of his brother-in-law H. Smith had the following peculiar property: The sum of the digits of that number was equal to the sum of the digits of the prime factors of that number. Got it? Smith's telephone number was 493-7775. This number can be written as the product of its prime factors in the following way:

$$4937775 = 3 \cdot 5 \cdot 5 \cdot 65837$$

The sum of all digits of the telephone number is  $4+9+3+7+7+7+5=42$ , and the sum of the digits of its prime factors is equally  $3+5+5+6+5+8+3+7=42$ . Wilansky was so amazed by his discovery that he named this kind of numbers after his brother-in-law: Smith numbers.

As this observation is also true for every prime number, Wilansky decided later that a (simple and unsophisticated) prime number is not worth being a Smith number, so he excluded them from the definition.

Wilansky published an article about Smith numbers in the Two Year College Mathematics Journal and was able to present a whole collection of different Smith numbers: For example, 9985 is a Smith number and so is 6036. However, Wilansky was not able to find a Smith number that was larger than the telephone number of his brother-in-law. It is your task to find Smith numbers that are larger than 4937775!

## Input

The input consists of a sequence of positive integers, one integer per line. Each integer will have at most 8 digits. The input is terminated by a line containing the number 0.

## Output

For every number  $n>0$  in the input, you are to compute the smallest Smith number which is larger than  $n$ , and print it on a line by itself. You can assume that such a number exists.

## Sample Input

```
4937774
0
```

---

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1133>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1142>

[3] <http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2203>

## Sample Output

4937775

## Problem Source

Mid-Central European Regional Contest 2000

### 【题目大意】

1982 年，里海大学（Lehigh University）的数学家 Albert Wilansky，在浏览电话簿时发现他姊夫 H.Smith 的电话号码有个奇怪特性：电话号码各位数字的和等于该数的质因子各位数字的和。Smith 的电话号码是 493-7775。该数可以如下方式表达：

$$4937775 = 3 \cdot 5 \cdot 5 \cdot 65837$$

电话号码的所有数字的和是  $4+9+3+7+7+7+5=42$ ，其质因子的各位数字的和是  $3+5+5+6+5+8+3+7=42$ 。于是 Wilansky 就将这惊奇的发现，以他姊夫的名字命名：Smith 数。

因为所有素数都有这个特性，Wilansky 后来定义素数不算 Smith 数，所以他把素数排除在定义之外。

在杂志 Two Year College Mathematics 上，Wilansky 发表了一篇关于 Smith 数的论文，并列举了所有的 Smith 数：例如，9985 是一个 Smith 数，6036 也是。但是，Wilansky 无法找到比他姊夫的电话号码大的 Smith 数。编程任务：找出大于 4937775 的 Smith 数！

### 输入格式

输入一些正整数，每行一个。每个整数最多有 8 位数字，当一行是 0 时，输入结束。

### 输出格式

对输入中每个  $n>0$  的数，计算并输出比  $n$  大的最小 Smith 数，占一行。假设 Smith 数存在。

### 【算法分析】

关于 Smith 数的资料，在网站上有很多，推荐阅读：

[http://en.wikipedia.org/wiki/Smith\\_number](http://en.wikipedia.org/wiki/Smith_number)

[http://www.experiencefestival.com/smith\\_number](http://www.experiencefestival.com/smith_number)

本题是输入一个正整数  $n$ ，要求计算比  $n$  大的最小 Smith 数。比较简单的办法就是从  $n$  开始，逐个数字进行判断，一直到找到了一个 Smith 数为止。判断一个数是否是 Smith 数，是按照 Smith 数的定义直接模拟，运行速度不快但方法简单。

(1) 求正整数  $n$  的各位数字的和是由如下函数实现的：

```
int digit(int n);
```

(2) 判断一个数  $n$  是否是素数，由如下函数实现：

```
int isPrime(int n);
```

函数中先把 2 的倍数去掉，然后整除到  $\sqrt{n}$ ，以便加快判断速度。

(3) 判断 Smith 数是由如下函数实现的：

```
int smith(int n);
```

判断 Smith 数时，用到质因子，这也只能逐个数字计算。注意函数中循环变量  $i++$  的位置和 continue 语句的使用，表示质因子可以是重复的，如题目中的数 4937775，质因子

中有两个 5。

### 【程序代码】

---

程序名称:	zju1133.c
题    目:	Smith Numbers
提交语言:	C
运行时间:	10ms
运行内存:	160KB

---

```
#include<stdio.h>
#include<math.h>

//求正整数 n 的各位数字的和
int digit(int n)
{
    int sum = 0;
    while(n)
    {
        sum += n%10;
        n/=10;
    }
    return sum;
}

//判断正整数 n 是否是素数
int isPrime(int n)
{
    if (n==2) return 1;
    if (n%2==0) return 0;           //将偶数去掉
    int k = (int)sqrt(n);           //计算到平方根就可以
    int i;
    for(i=3; i<=k; i+=2)
        if(n%i==0)
            return 0;
    return 1;
}

//判断 Smith 数
int smith(int n)
{

```



```

int i;
int temp = n;
int sum = 0;
//这里求素数时，采用的是筛法
int k = (int)sqrt(n); //上限
for(i=2; i<=k;){
    if(n%i==0)
    {
        sum += digit(i); //i 是素数，累加各位数字的和
        n/=i;
        k = (int)sqrt(n); //不断调整上限
        continue; //质因子可以是重复的
    }
    i++; //注意：不能放在 for 语句中
}
//最后的商也是质因子
sum += digit(n);
//是否是 Smith 数
if (sum==digit(temp)) return 1;
return 0;
}

int main()
{
    int n;
    while(scanf("%d",&n) && n)
    {
        //从 n 开始，逐个数字计算，直到找到 Smith 数
        while(n++)
            if(!isPrime(n) && smith(n)) break;
        printf("%d\n",n);
    }
    return 0;
}

```

# ZJU1158-Treasure Hunt<sup>[1, 2, 3]</sup>

Time Limit: 10 Seconds

Memory Limit: 32768KB

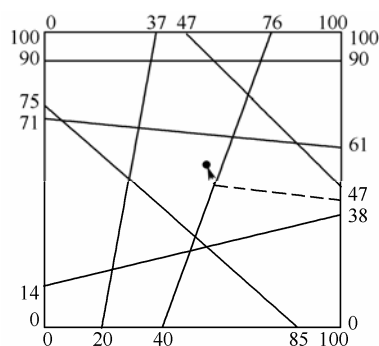
Archeologists from the Antiquities and Curios Museum (ACM) have flown to Egypt to examine the great pyramid of Key-Ops. Using state-of-the-art technology they are able to determine that the lower floor of the pyramid is constructed from a series of straightline walls, which intersect to form numerous enclosed chambers. Currently, no doors exist to allow access to any chamber. This state-of-the-art technology has also pinpointed the location of the treasure room. What these dedicated (and greedy) archeologists want to do is blast doors through the walls to get to the treasure room. However, to minimize the damage to the artwork in the intervening chambers (and stay under their government grant for dynamite) they want to blast through the minimum number of doors. For structural integrity purposes, doors should only be blasted at the midpoint of the wall of the room being entered. You are to write a program which determines this minimum number of doors.

An example is shown below (Fig):

This problem contains multiple test cases!

The first line of a multiple input is an integer  $N$ , then a blank line followed by  $N$  input blocks. Each input block is in the format indicated in the problem description. There is a blank line between input blocks.

The output format consists of  $N$  output blocks. There is a blank line between output blocks.



## Input

The input will consist of one case. The first line will be an integer  $n$  ( $0 \leq n \leq 30$ ) specifying number of interior walls, followed by  $n$  lines containing integer endpoints of each wall  $x_1 \ y_1 \ x_2 \ y_2$ . The 4 enclosing walls of the pyramid have fixed endpoints at (0,0), (0,100), (100,100) and (100,0) and are not included in the list of walls. The interior walls always span from one exterior wall to another exterior wall and are arranged such that no more than two walls intersect at any point. You may assume that no two given walls coincide. After the listing of the interior walls there will be one final line containing the floating point coordinates of the treasure in the treasure room (guaranteed not to lie on a wall).

[1] <http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1158>

[2] <http://acm.pku.edu.cn/JudgeOnline/problem?id=1066>

[3] <http://online-judge.uva.es/p/v7/754.html>

## Output

Print a single line listing the minimum number of doors which need to be created, in the format shown below.

## Sample Input

```
1
7
20 0 37 100
40 0 76 100
85 0 0 75
100 90 0 90
0 71 100 61
0 14 100 38
100 47 47 100
54.5 55.4
```

## Sample Output

```
Number of doors = 2
```

## Problem Source

East Central North America 1999

### 【题目大意】

来自文物和古董博物馆（ACM）的考古学家，已飞往埃及考察伟大的胡夫金字塔<sup>[1, 2, 3]</sup>（也称为Pyramid of Khufu, Great Pyramid of Giza, Pyramid of Cheops, 发音: "key-ops"）。利用最先进的技术，他们能够确定金字塔的底层是由一系列的直墙所建造而成的，它们相交形成许多封闭隔间。一般情况下，任何一个隔间都没有门。这种最先进的技术，已经能够精确的确定宝藏室的位置。这些专注（和贪婪）的考古学家想要做的事，就是在墙上爆炸出门而进入宝藏室。然而，为了尽量减少对连接隔间艺术墙的破坏（且在政府的授权下使用炸药），他们想爆炸最少数量的门。为了结构的完整性，门应该位于要进入隔间的墙中间。编程任务，确定门的最小数量。

本题包含多组测试例！

多组测试例的第一行是一个整数  $N$ ，然后是一个空行，接着是  $N$  个输入数据块。每个数据块的格式在问题描述中给出。每个数据块之间有一个空行。

输出格式包括  $N$  个输出数据块，每个输出数据块之间有一个空行。

### 输入格式

输入只有一个测试例（指每个数据块内）。第一行是一个整数  $n$  ( $0 \leq n \leq 30$ )，表示内部墙的数量。接下来  $n$  行，是每个墙的端点坐标:  $x_1 y_1 x_2 y_2$ （整数）。金字塔的 4 个封闭围墙，

---

[1] [http://en.wikipedia.org/wiki/Great\\_Pyramid\\_of\\_Giza](http://en.wikipedia.org/wiki/Great_Pyramid_of_Giza)

[2] <http://interoz.com/EGYPT/cheops.htm>

[3] <http://baike.baidu.com/view/3924.htm>

其端点坐标是固定的：(0,0), (0,100), (100,100) 和 (100,0)，内部墙中不会有这些坐标。内部墙总是从一个外墙跨到另一个外墙，并且在任何一点，只有两堵墙壁相交。你可以假设，没有两堵墙壁是重叠的。在内部墙의列表之后，还有最后一行，是在宝藏室中宝藏的浮点坐标（保证不会位于墙上）。

（样例数据就是题目中的插图）

**输出格式**

输出一行，是需要爆炸的门的最小数量，格式如样例所示。

**【算法分析】**

在边长为 100 的正方形内，由很多端点在正方形边上的线段围成封闭的空间。宝藏在某一个封闭空间内，要求从正方形外进入该空间而经过的线段数最少。在 East Central North America 1999 的比赛网站（<http://plg1.cs.uwaterloo.ca/~acm00/regionals/>）上有标程，这些程序都比较复杂。

有一个直观且简单的办法判断是否需要开一个门，如图 9-27 所示。

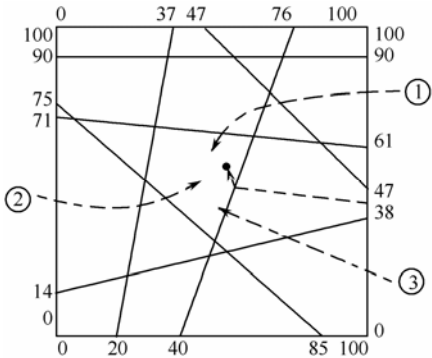


图 9-27 门与宝藏的关系

对任意一个内墙，如果门与宝藏在该墙的同侧，就不需要开门；否则，需要开一个门。图 9-26 中，在外墙①处开门，有三堵墙使门与宝藏异侧，需要开 4 个门；在外墙②或者③处开门，都有两堵墙使门与宝藏异侧，需要开三个门。

**(1) 数据结构**

内墙端点坐标的结构体和数组：

```
struct interior
{
    int x1, y1, x2, y2;
}inner[40];
```

所有墙（内墙和外墙）的端点坐标的结构体和数组：

```
struct exterior
{
    int x, y;
}outer[40];
```

内部墙的数量：

```
int n;
```

宝藏的位置坐标：

```
double tx, ty;
```

(2) 判断宝藏与每一堵墙的位置关系

点与直线的位置关系，如图 9-28 所示。

直线方向是由  $A(x_1, y_1)$  到  $B(x_2, y_2)$  的方向，直线方程为

$$ax+by+c=0$$

将两点的坐标代入，解得

$$a=y_2-y_1;$$

$$b=x_1-x_2;$$

$$c=x_2 \times y_1 - x_1 \times y_2;$$

这三个参数的计算，由函数完成：

```
void judge(struct interior w, struct sides *e)
```

形参  $w$  是当前计算的内墙，形参  $e$ （按地址引用）保存计算结果。

点与直线的位置关系：

```
side = a*x+b*y+c
```

若  $side < 0$ ，则点  $P(x, y)$  在直线的左侧；若  $side > 0$ ，则点  $P$  在直线的右侧；若  $side = 0$ ，则点  $P$  在直线上（题目中确保宝藏不会位于墙上）。

宝藏与每一堵墙的位置关系，保存到数组中：

```
int s1[40];
```

(3) 计算需要爆炸的门的数量

在外墙的任意两个内墙之间（包括外墙的墙角），都可以开门。但是，这两个内墙的端点必须位于同一个外墙上。如何判断两个内墙的端点位于同一个外墙上？方法是枚举。如果任意两个内墙的端点坐标：

```
x1 = outer[i].x+outer[j].x;
```

```
y1 = outer[i].y+outer[j].y;
```

则  $x_1=0$ ，是左边的墙； $x_1=200$ ，是右边的墙； $y_1=0$ ，是底边的墙； $y_1=200$ ，是顶边的墙。然后利用图 9-26 中门与宝藏的关系，判断外墙上的门、宝藏与每一个内墙的位置关系，如果门与宝藏在该墙的同侧，就不需要开门；否则，需要开一个门。累计门的数量，最后取最小的数量。

按理应该是  $(x_1/2, y_1/2)$  才是中点坐标，但这样就有一个精度问题。例如： $A(0, 1)$ ， $B(10, 0)$ ， $A$  在左边墙上， $B$  在底边墙上。求得中点坐标  $(5, 0)$ 。其中  $y_1=0$ ，误以为两个端点都在底边墙上。所以在上述判断中，是把顶边和右边墙坐标变成 200。

输出时，还要加上外墙上的一个门。

## 【程序代码】

程序名称：	zju1158.c
题 目：	Treasure Hunt
提交语言：	C
运行时间：	0ms
运行内存：	160KB

```
#include <stdio.h>
```

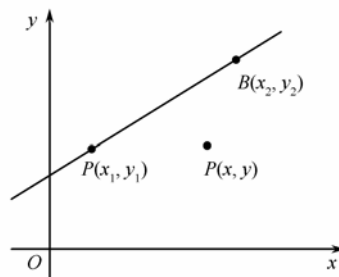


图 9-28 点与直线的位置关系

```

#define ZERO 1e-4

//内墙端点坐标的结构体和数组
struct interior
{
    int x1, y1, x2, y2;
}inner[40];

//判断宝藏与内墙位置的结构体和数组
struct sides
{
    int a, b, c;
}side[40];

//所有墙（内墙和外墙）的端点坐标的结构体和数组
struct exterior
{
    int x, y;
}outer[40];

//判断宝藏与内墙位置
void judge(struct interior w, struct sides *e)
{
    e->a = w.y2-w.y1;
    e->b = w.x1-w.x2;
    e->c = w.x2*w.y1-w.x1*w.y2;
}

int main()
{
    int i, j, k;
    int x1, y1, x2, y2;           //墙的端点坐标
    int n;                         //内部墙的数量
    double tx, ty;                 //宝藏的位置坐标
    int sl[40];                    //宝藏与每一堵墙的位置关系

    int N;                         //数据块的数量
    scanf("%d", &N);
    while (N--)
    {
        scanf("%d", &n);
        //外墙的端点坐标是固定的: (0,0), (0,100), (100,100) 和 (100,0)
        outer[0].x = 0;
        outer[0].y = 0;
        outer[1].x = 0;
        outer[1].y = 100;
    }
}

```

```

outer[2].x = 100;
outer[2].y = 100;
outer[3].x = 100;
outer[3].y = 0;
int sp = 3;
//读取所有内墙的端点坐标
for(i=0; i<n; i++)
{
    scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
    outer[++sp].x = x1;           //墙的端点坐标
    outer[sp].y = y1;
    outer[++sp].x = x2;
    outer[sp].y = y2;
    inner[i].x1 = x1;           //内墙的端点坐标
    inner[i].y1 = y1;
    inner[i].x2 = x2;
    inner[i].y2 = y2;
}
//读取宝藏的端点坐标
scanf("%lf%lf", &tx, &ty);
//判断宝藏与每一堵墙的位置关系
double side0;
for(i=0; i<n; i++)
{
    judge(inner[i], &side[i]);
    side0 = side[i].a*tx+side[i].b*ty+side[i].c;
    if (side0>ZERO) s1[i] = 1;
    else if(side0<ZERO) s1[i] = -1;
    else s1[i] = 0;
}
//计算需要爆炸的门的的最小数量
int door;
//∵门的数量小于n, ∴.9999 就是∞
int minDoor = 9999;
int sidel;
int s2;
//对任意的两个墙端点
for(i=0; i<sp; i++)
    for(j=i+1; j<=sp; j++)
    {
        x1 = outer[i].x+outer[j].x;
        y1 = outer[i].y+outer[j].y;
        //这两个墙端点位于某一侧外墙上
        if((x1==0)|| (y1==0)|| (x1==200)|| (y1==200))
        {
            //从这两个端点处开门, 到达宝藏时需要门的数量

```

```

door = 0;
//判断外墙上的这个门与每个内墙的位置关系
for(k=0; k<n; k++)
{
    side1 = side[k].a*x1+side[k].b*y1+2*side[k].c;
    if (side1>0) s2 = 1;
    else if(side1<0) s2 = -1;
    else
    {
        door = 9999; break;
    }
    //宝藏与外墙上的门，位于该墙的两侧，需要开一个门
    if (s2!=s1[k]) door++;
}
if (minDoor>door) minDoor = door;
}
}
//外墙本身也要开一个门
printf("Number of doors = %d\n", ++minDoor);
if (N) printf("\n");
}
return 0;
}

```



# 索引

ZJU1081-Points Within .....	291
ZJU1082-Stockbroker Grapevine .....	223
ZJU1083-Frame Stacking .....	228
ZJU1084- Channel Allocation .....	116
ZJU1085-Alien Security .....	120
ZJU1086-Octal Fractions .....	1
ZJU1088-System Overload .....	32
ZJU1089-Lotto .....	3
ZJU1090-The Circumference of the Circle .....	6
ZJU1091-Knight Moves .....	126
ZJU1092-Arbitrage .....	235
ZJU1093-Monkey and Banana .....	163
ZJU1094-Matrix Chain Multiplication .....	99
ZJU1095-Humble Numbers .....	8
ZJU1096-Subway .....	296
ZJU1097-Code the Tree .....	104
ZJU1098-Simple Computers .....	36
ZJU1099-HTML .....	11
ZJU1100-Mondriaan's Dream .....	169
ZJU1101-Gamblers .....	132
ZJU1102-Phylogenetic Trees Inherited .....	174
ZJU1103-Hike on a Graph .....	135
ZJU1104-Leaps Tall Buildings .....	301
ZJU1105-FatMouse's Tour .....	15
ZJU1107-FatMouse and Cheese .....	180
ZJU1108-FatMouse's Speed .....	183
ZJU1109-Language of FatMouse .....	70
ZJU1110-Dick and Jane .....	306
ZJU1111-Poker Hands .....	74
ZJU1112-Equidistance .....	309
ZJU1114-Problem Bee .....	315
ZJU1115-Digital Roots .....	17
ZJU1116-A Well-Formed Problem .....	81
ZJU1117-Entropy .....	239
ZJU1118-N-Credible Mazes .....	245

ZJU1119-SPF .....	251
ZJU1121-Reserve Bookshelf .....	40
ZJU1122-Clock .....	19
ZJU1123-Triangle Encapsulation.....	319
ZJU1125-Floating Point Numbers .....	325
ZJU1126-Bio-Informatics .....	88
ZJU1127-Roman Forts .....	256
ZJU1128-Atlantis .....	330
ZJU1129-Erdos Numbers.....	141
ZJU1130-Ouroboros Snake.....	262
ZJU1132-Railroad.....	188
ZJU1133-Smith Numbers .....	335
ZJU1134-Strategic Game .....	268
ZJU1136-Multiple.....	147
ZJU1137-Girls and Boys.....	273
ZJU1139-Rectangles .....	22
ZJU1140-Courses.....	278
ZJU1141-Closest Common Ancestors .....	281
ZJU1142-Maze.....	152
ZJU1143-Date Bugs.....	48
ZJU1144-Robbery.....	51
ZJU1145-Dreisam Equations .....	207
ZJU1146-LC-Display.....	56
ZJU1147-Formatting Text.....	195
ZJU1148-The Game.....	158
ZJU1149-Dividing .....	202
ZJU1150-S-Trees .....	285
ZJU1151-Word Reversal.....	25
ZJU1152-A Mathematical Curiosity .....	27
ZJU1153-Tournament Seeding .....	61
ZJU1154-Niven Numbers .....	29
ZJU1156-Unscrambling Images .....	109
ZJU1157-A Plug for UNIX.....	214
ZJU1158-Treasure Hunt.....	339
ZJU1159-487-3279 .....	93

## 参考文献

- [1] 王晓东. 计算机算法设计与分析 (第 3 版). 北京: 电子工业出版社, 2007.
- [2] <http://www.acm.inf.ethz.ch/PSArchiveChanges.html>
- [3] (美) John A. Dossey, Albert D. Otto, 等. 离散数学 (原书第 5 版). 章炯民等译. 北京: 机械工业出版社, 2007.
- [4] (美) Kenneth H. Rosen, 离散数学及其应用 (原书第 5 版). 袁崇义, 等译. 北京: 机械工业出版社, 2007
- [5] 刘汝佳, 黄亮. 算法艺术与信息学竞赛. 北京: 清华大学出版社, 2004
- [6] <http://www.informatik.uni-ulm.de/acm/Locals/1996/>
- [7] (德) Nicolai M. Josuttis 著, C++标准程序库. 侯捷, 孟岩译. 武汉: 华中科技大学出版社, 2002.9
- [8] 严蔚敏, 吴伟明, 等. 数据结构, 北京: 清华大学出版社, 2004.9